Course on Data Analysis and Visualization

Leif Kobbelt, Torsten Kuhlen, Isaak Lim,

Christian Nowke, Andrea Schnorr, **Benjamin Weyers**

Visual Computing Institute

RWTH AACHEN UNIVERSITY

The responsible groups

RWTH

Faculty of Mathematics, Comp. Science and Nat. Sciences

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Virtual Reality and Immersive Visualization
Prof. Torsten W. Kuhlen

Computer Graphics and Multimedia
Prof. Leif Kobbelt

IT Center | RWTH AACHEN UNIVERSITY

Central Institutions

How To Find Us - Virtual Reality and Immersive Visualization

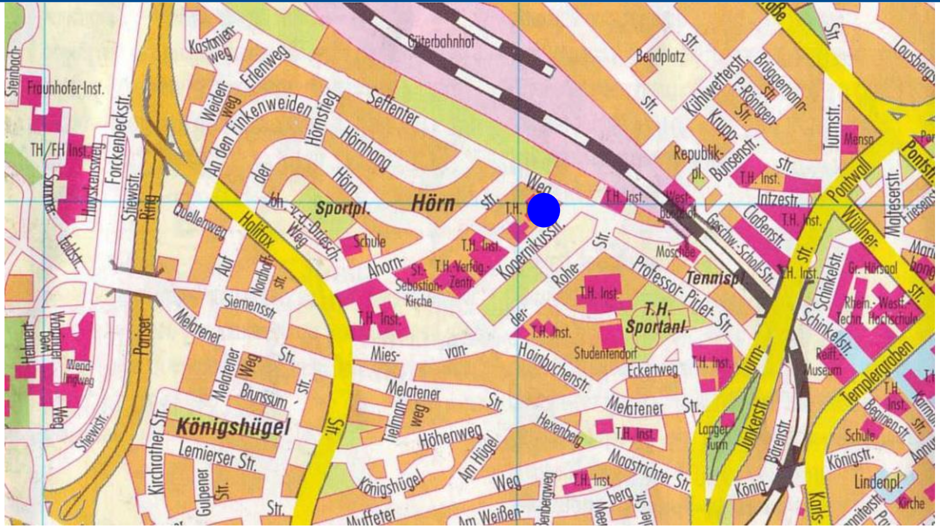
IT Center

Kopernikusstraße 6 (Erweiterungsbau)

Email: **weyers@vr.rwth-aachen.de**

{nowke, schnorr}@vr.rwth-aachen.de

www.vci.rwth-aachen.de



How To Find Us - Computer Graphics and Multimedia

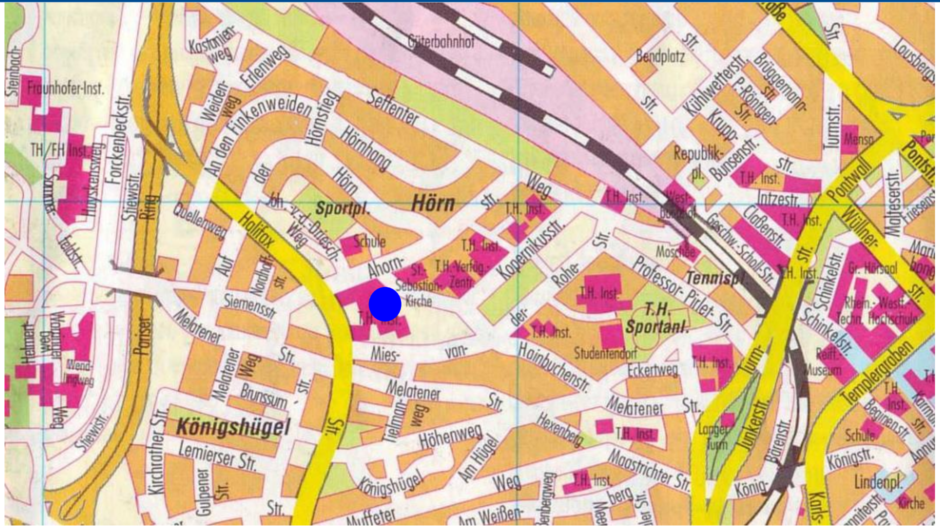
Informatik E3

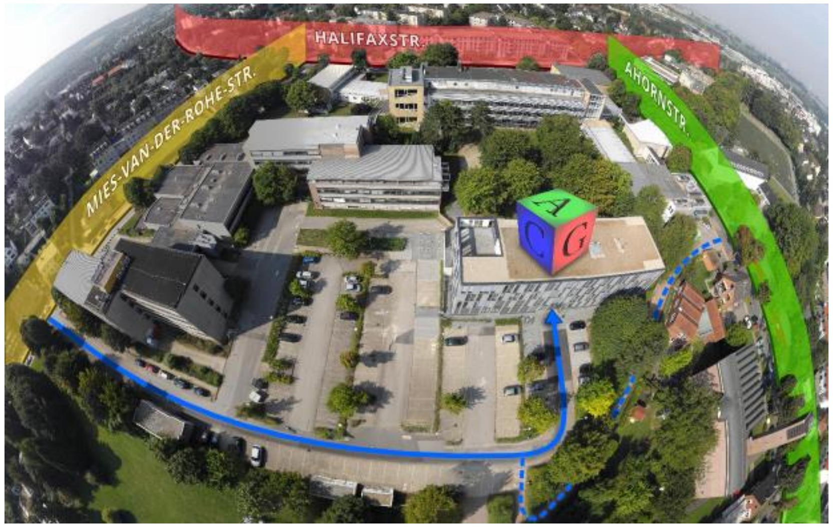
Ahornstr.

Email: **kobbelt@informatik.rwth-aachen.de**

issak.lim@cs.rwth-aachen.de

www.vci.rwth-aachen.de





Who are your Teachers in this Course?

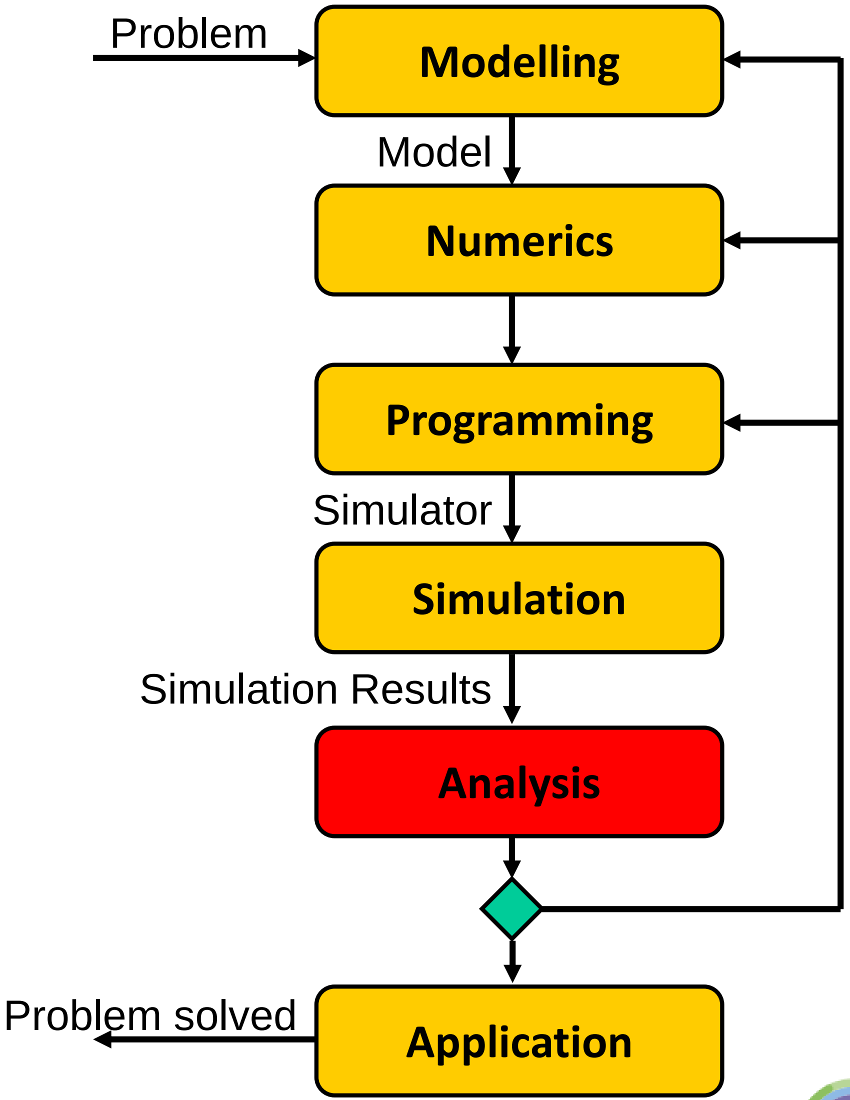
- Computer Graphics & Multimedia
  - Prof. Dr. Leif Kobbelt (Lecture)
    - *Scientific Visualization*, Scalar and Vector Field Visualization, Data Types, Grid Interpolation
  - Isaak Lim, M.Sc.
    - Exercises

- Virtual Reality & Immersive Visualization
  - Dr. Benjamin Weyers (Lecture)
    - *Information Visualization*, Rendering, Virtual Reality, Perception
  - Dipl.-Inform. Christian Nowke
  - Andrea Schnorr, M.Sc.
    - Exercises

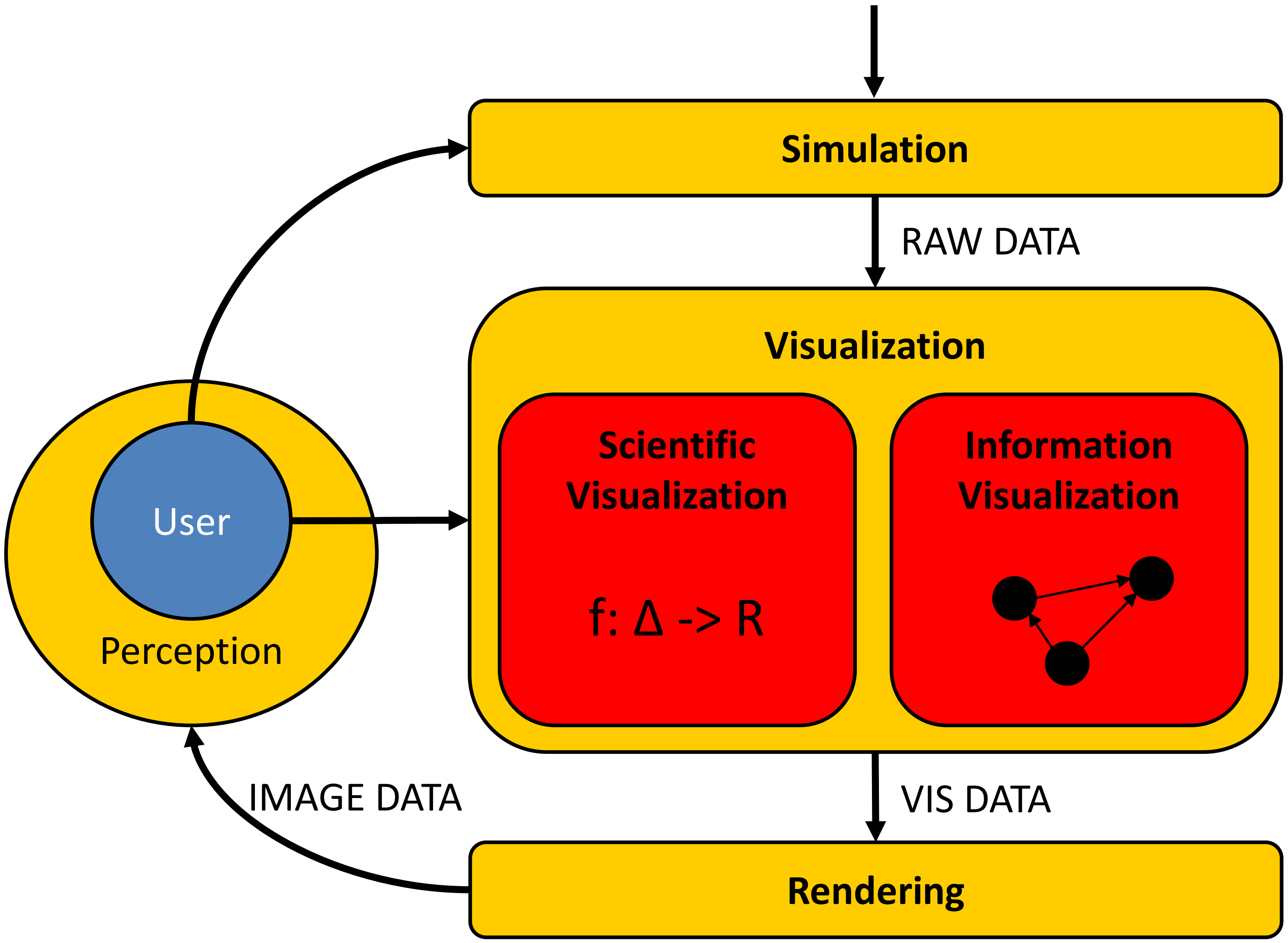
Who Should Attend this Course?

- Computational Engineering Science (B.Sc.)
  - 5. Semester, Pflichtfächer
- Simulation Sciences (M.Sc.)
  - Mandatory Courses, Semester 1

The Simulation Loop

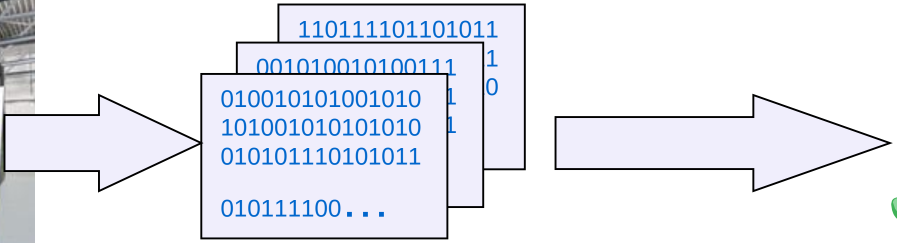


Closer look to (Visual) Analysis in the Simulation Loop

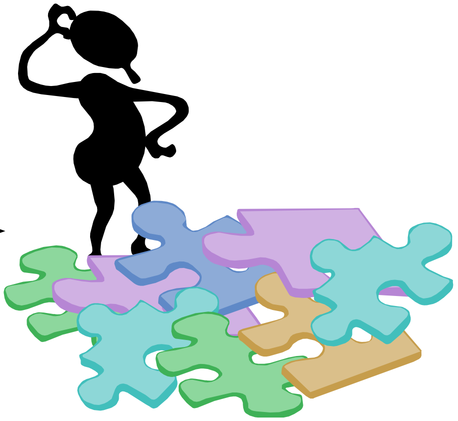


Why Data Analysis & Visualization?

JUQUEEN IBM Blue Gene/Q System, JSC, Forschungszentrum Jülich GmbH



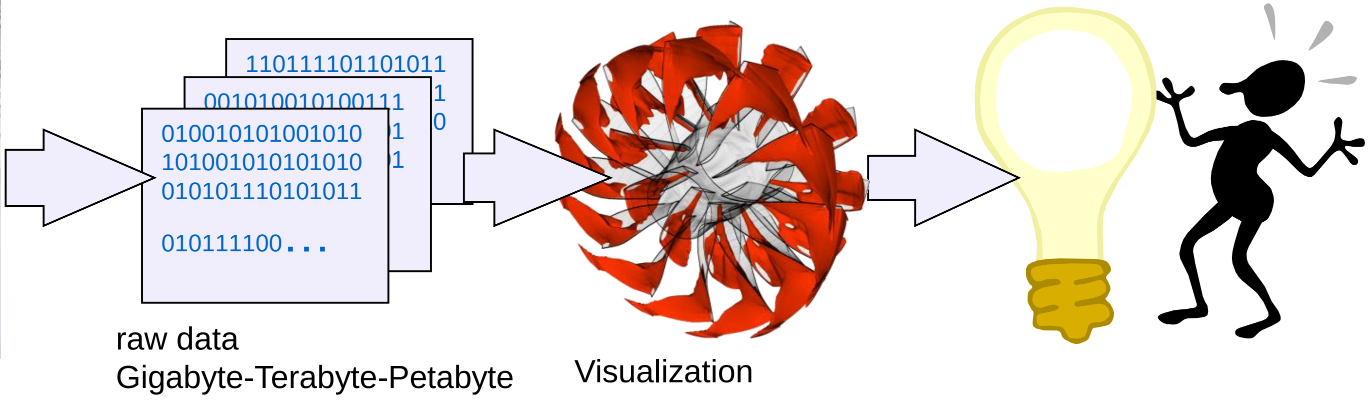
raw data
Gigabyte-Terabyte-Petabyte



- Generating insight from data requires data analysis.
  Hamming: „The purpose of Computing is insight, not numbers!“

Why Data Analysis & Visualization?

JUQUEEN IBM Blue Gene/Q System, JSC, Forschungszentrum Jülich GmbH



- Generating insight from data requires data analysis.
  Hamming: „The purpose of Computing is insight, not numbers!“

- Visualization is (primarily) being used for the analysis process

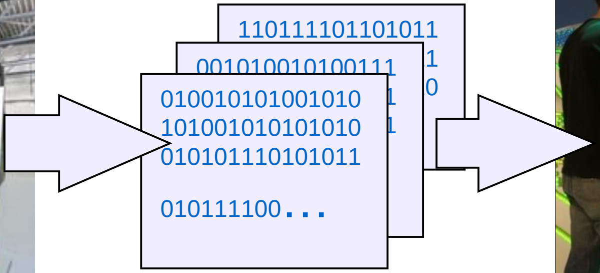
EU Flagship: The Human Brain Project

- 1 Billion Euros Funding for the next 10 years
- A major goal:
  - Simulation of large biologically realistic neural networks
  - … up to the human brain scale (10^{11} neurons, 10^{15} synapses)
  - Extremely dynamic data at high temporal resolution

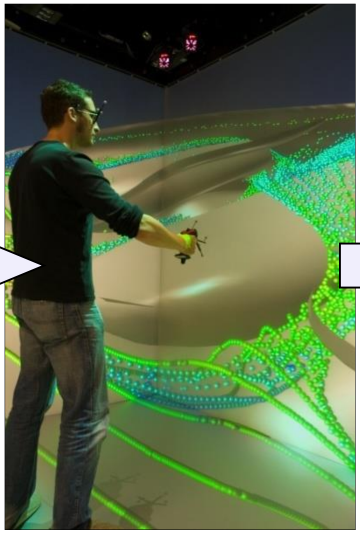
[Courtesy of José M. Peña,
Universidad Politécnica de Madrid]

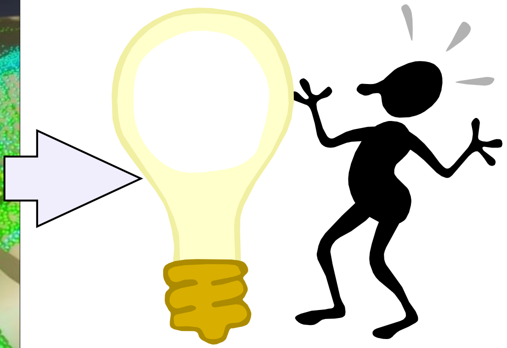
Why Data Analysis & Visualization?

JUQUEEN IBM Blue Gene/Q System, JSC,
Forschungszentrum Jülich GmbH



raw data
Gigabyte-Terabyte-Petabyte

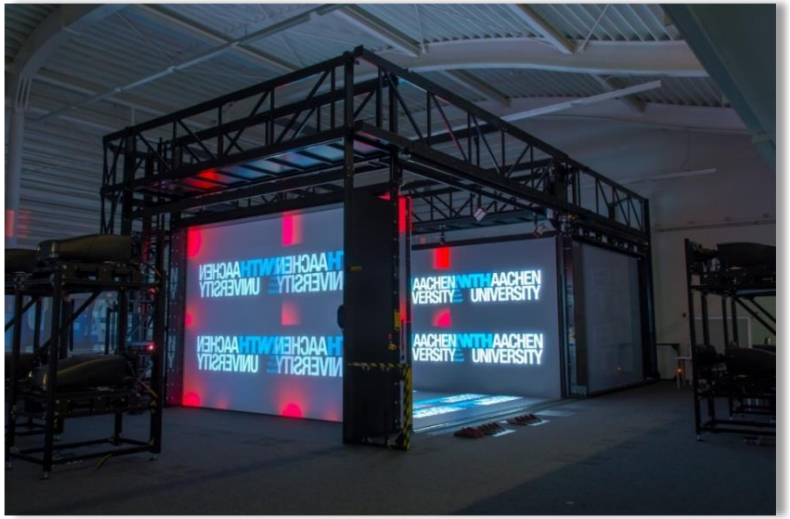


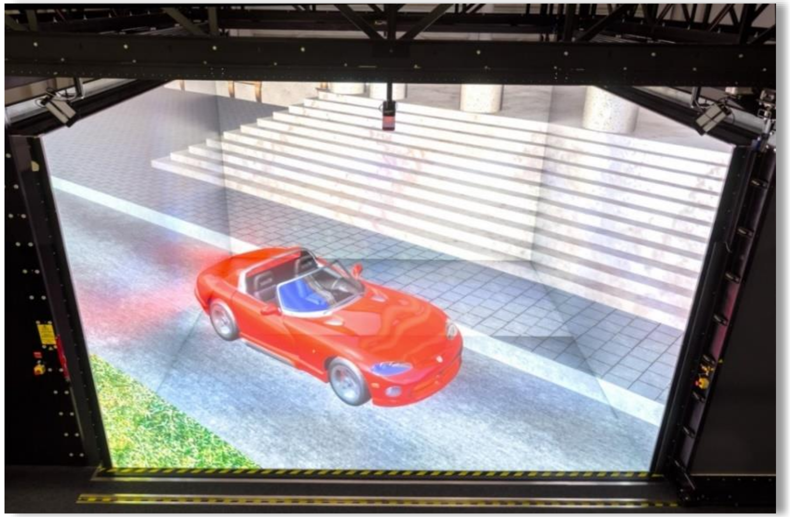


- Generating insight from data requires data analysis.
  Hamming: „The purpose of Computing is insight, not numbers!“

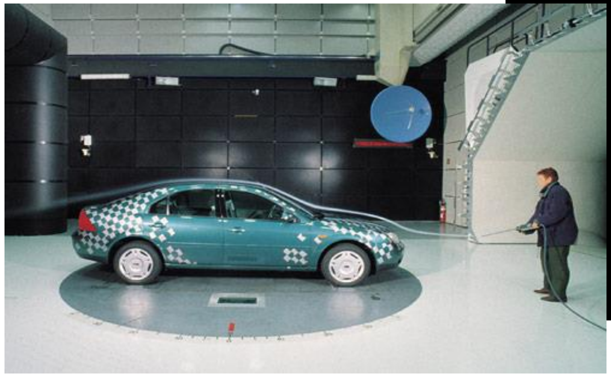
- Visualization is (primarily) being used for the analysis process
- Amount of raw data is rapidly increasing: Finer grids, 3-D, time-variant
- Explorative versus confirmative analysis, Virtual Reality

The aixCAVE @ RWTH



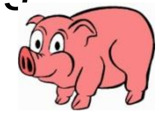


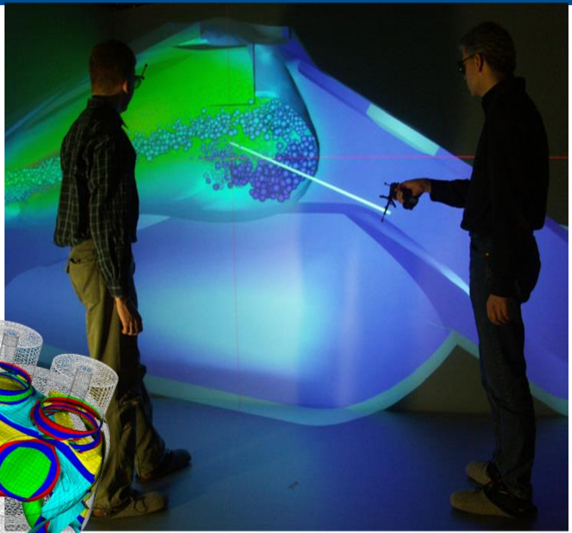
The “Virtual Windtunnel”

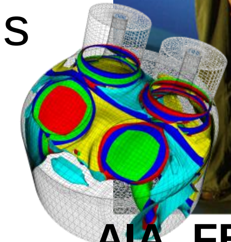


VR Applications in CES @ RWTH

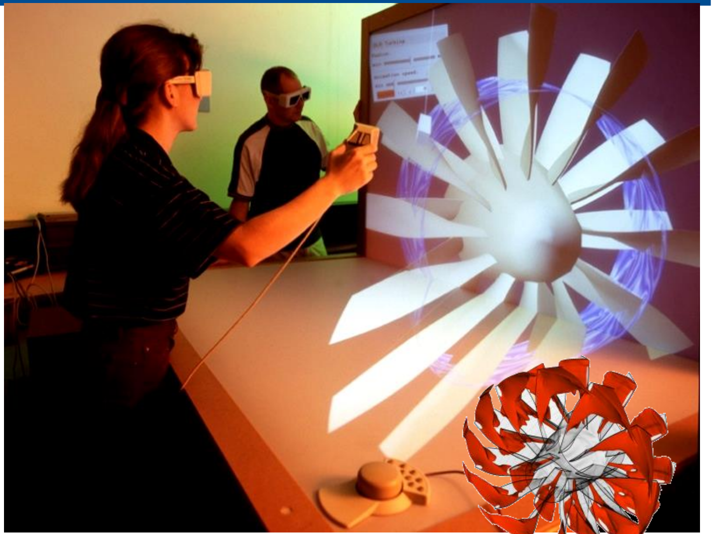
- Motors & Turbines
- Twin Extruders
- Nasal Airflow
- Bood Flow
- Crash Simulations
- Material Science
- Pig housing



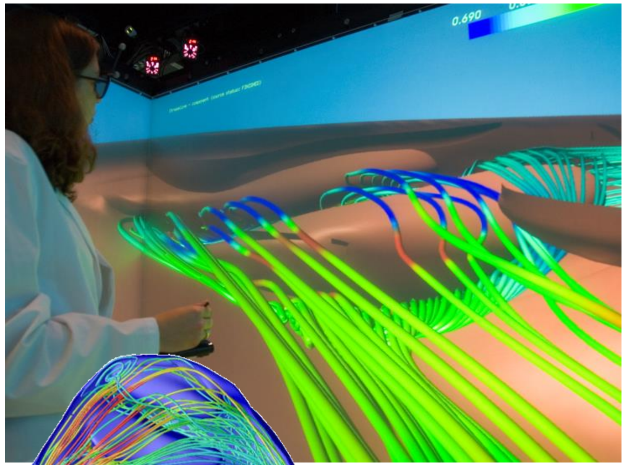


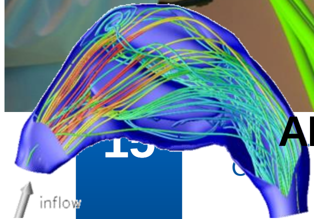


AIA, FEV GmbH, VR Group

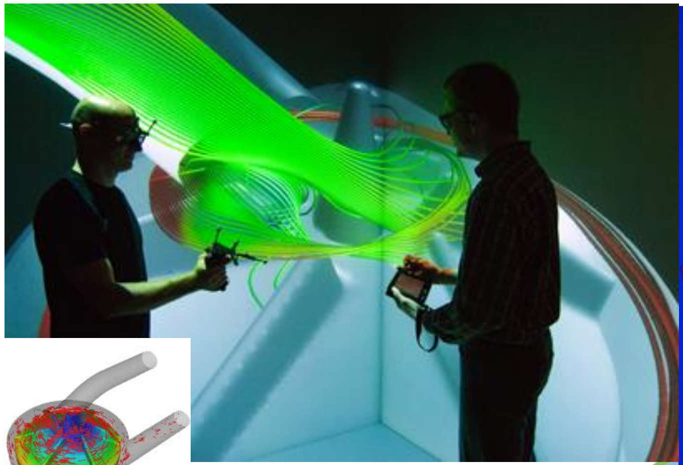


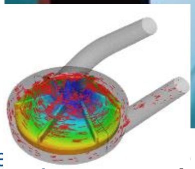
DLR, IST, VR Group



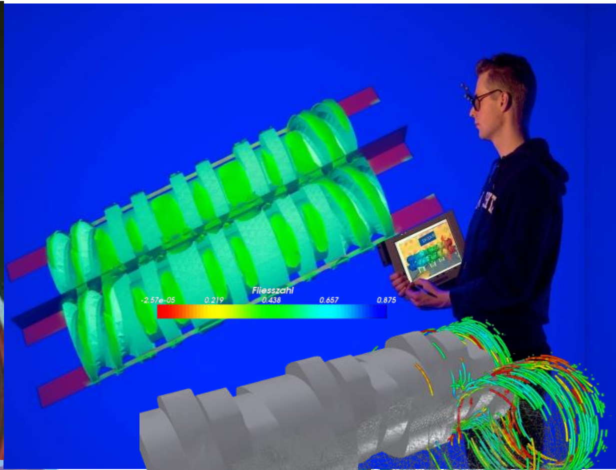


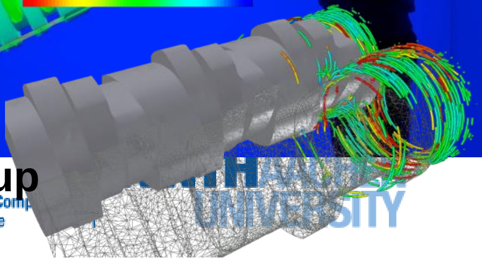
AIA, VR Group





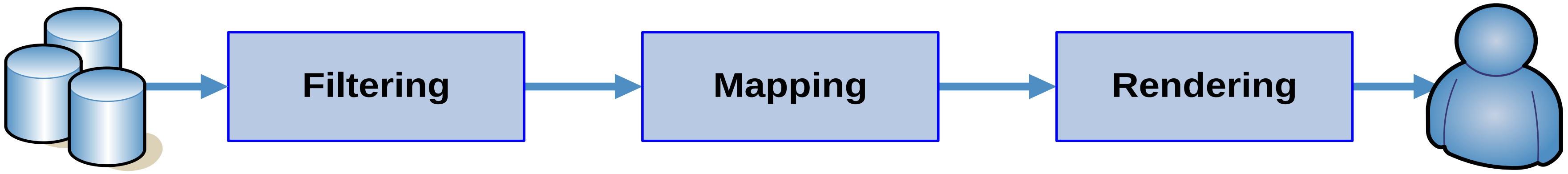
VR Group





IKV, VR Group

The Visualization Pipeline



- **Filtering** = Preprocessing of data
  e.g. subsampling, dimension reduction…
- **Mapping** = Transformation to “primitives” (graphical objects)
  e.g. polygonal meshes
- **Rendering** = Transformation of render-primitives
  to output images

Course Outline

20.10.2015

(V1) Introduction & Basics - Rendering (Weyers)

Course Introduction, Organization, Visualization and Rendering Pipeline

27.10.2015

(V2) Basics - Rendering (Nowke)

Shading, Scan Conversion, Texture Mapping

03.11.2015

(Ü1) (Schnorr/Nowke)

(V3) Basics - Perception (Weyers)

Sensory and Abitrary Perception, Human Eye, Color Models, Gestalt Psychology

10.11.2015

(Ü2) (Schnorr/Nowke)

(V4) Basics - Visualization (Kobbelt)

Data types, grid interpolation and integration, & Scalar field visualization: transfer function design, iso-contouring, volume rendering

Course Outline (cont.)

17.11.2015

(Ü3) (Lim)

(V5) Scientific Visualization (Kobbelt)

Vector field visualization: glyphs, flow lines, streak lines

24.11.2015

(Ü4) (Lim)

(V6) Large Data Analysis (Kobbelt)

01.12.2015

(Ü5) (Lim)

(V7) Large Data Analysis (Kobbelt)

08.12.2015

(Ü6) (Lim)

(V8) Immersive Visualization (Weyers)

Explorative analysis of simulation datasets in Virtual Reality

Course Outline (cont.)

15.12.2015
(Ü7) (Schnorr/Nowke)
(V9) Information Visualization (Weyers)
Introduction to information visualization and Data Types

22.12.2015 --- NO COURSES ---

12.01.2016
(Ü8) (Schnorr/Nowke)
(V10) Information Visualization (Weyers)
Visualization and representation of values and relations

19.01.2016
(Ü9) (Schnorr/Nowke)
(V11) Information Visualization (Weyers)
Visualization of graphs and node link diagrams

Course Outline (cont.)

26.01.2016

(Ü10) (Schnorr/Nowke)

(V12) Information Visualization (Weyers)

Graph drawing and layout algorithms

02.02.2016

(Ü11) (Schnorr/Nowke)

(V13) Information Visualization (Weyers)

Graph drawing and layout algorithms, limitations of information visualization

09.02.2016

(Ü12) (Schnorr/Nowke)

Questions and Answers (Lim/Schnorr/Nowke/Weyers)

Exams – Check in Campus!

First Exam: Thursday, 18.02.2016, 13:15 to 15:30 (AH V)

Second Exam: Tuesday, 22.03.2016, 10:00 to 12:00 (5056)

Organizational Stuff

- Literature: „Handapparat“ in Computer Science Library
    - Script? Course contents, e.g., in A. Watt, CG, und K.-F- Kraiss, AMMI
- Exercises: Just BEFORE the lessons
- Material: L2P (Exercise and Lecture)
- Miscellaneous, announcements, exercises: L2P

- Relevant for Exam:
    - Content presented in lecture
    - Content presented and discussed in exercises!

Literature for the Rendering / Virtual Reality Part

Books (*Handapparat Informatik-Bibliothek):

- *D. Bowman et al. 3D User Interfaces. Addison-Wesley
- *K. M. Stanney. Handbook of Virtual Environments. Erlbaum
- *M.Slater et al. Computer Graphics & Virtual Environments. Addison-Wesley
- *G. Burdea, P. Coiffet. Virtual Reality Technology. John Wiley & Sons
- *K.-F. Kraiss (Ed.). Advanced Man Machine Interfaces. Springer
- R.S. Kalawski. The Science of Virtual Reality and Virtual Environments. Addison Wesley
- F. Dai. Lebendige virtuelle Welten. Springer Verlag
- J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes. Computer Graphics – Principles and Practice. Addison Wesley
- K.D. Tönnies, H.U. Lemke. 3D-Computergrafische Darstellungen. Oldenburg Verlag
- *A. Watt. 3D Computer Graphics. Addison Wesley

- **Conferences:**
  - IEEE VR, IEEE Vis
  - IPT, EGVE
  - ACM Conferences (SIGGRAPH, VRST, VRCAI, ...)

Course on Data Analysis and Visualization

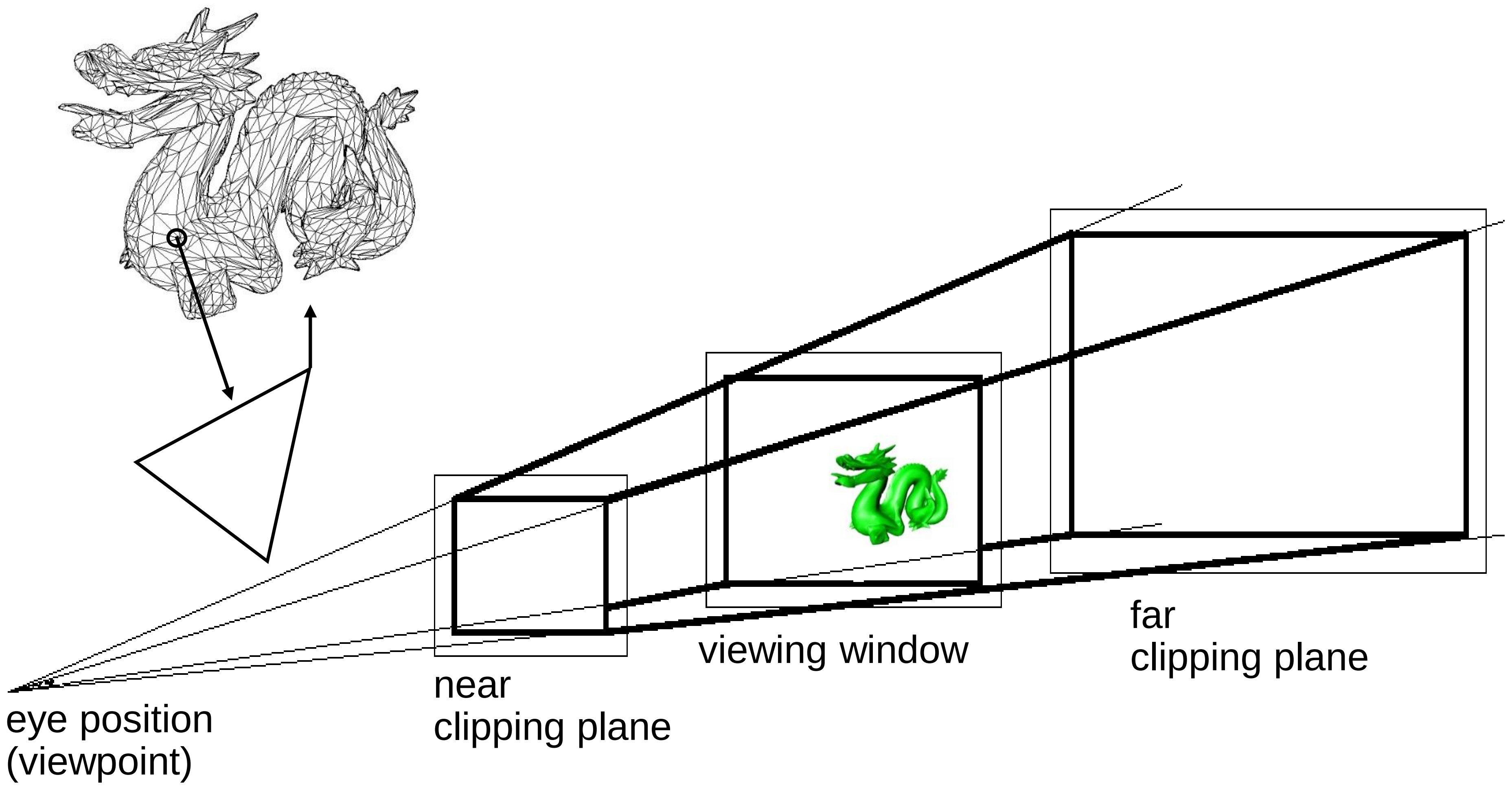
Basics in Computer Graphics
The Rendering Pipeline

-- Benjamin Weyers & Christian Nowke --

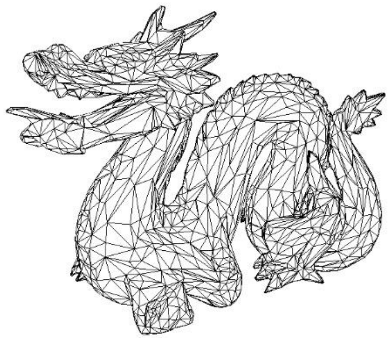
Topics – Basics in Computer Graphics / Rendering Pipeline

- **Rendering Pipeline**
- **Representation of rigid objects**
- **Transformations**
- **Culling**
- **Projection**
- **Clipping**
- **Hidden Surface Removal**
- **Shading**
- **Scan Conversion**
- **Texture Mapping**
- **Graphics Hardware**
- **Ray Tracing *(optional topic)***

View Volume


eye position
(viewpoint)
near
clipping plane
viewing window
far
clipping plane

Overview – Rendering Pipeline



Transformation:
Position und orientation
of objects

Transformation:
WKS, SKS

Culling

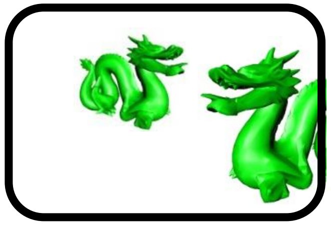
Projection

Clipping

Shading

Hidden Surface
Removal

Scan
Conversion



Steps – Rendering Pipeline

1. **Transformations:**
   – Describe position & orientation of objects within a scene
   – Transform local coordinates of objects into a single world coordinate system (WKS)
   – Optional: Transform into a view Coordinate system (SKS): viewpoint = origin
2. **Culling:**
   – Remove invisible polygons
3. **(Perspective) Projection:**
   – Transform Vertices into screen coordinate system

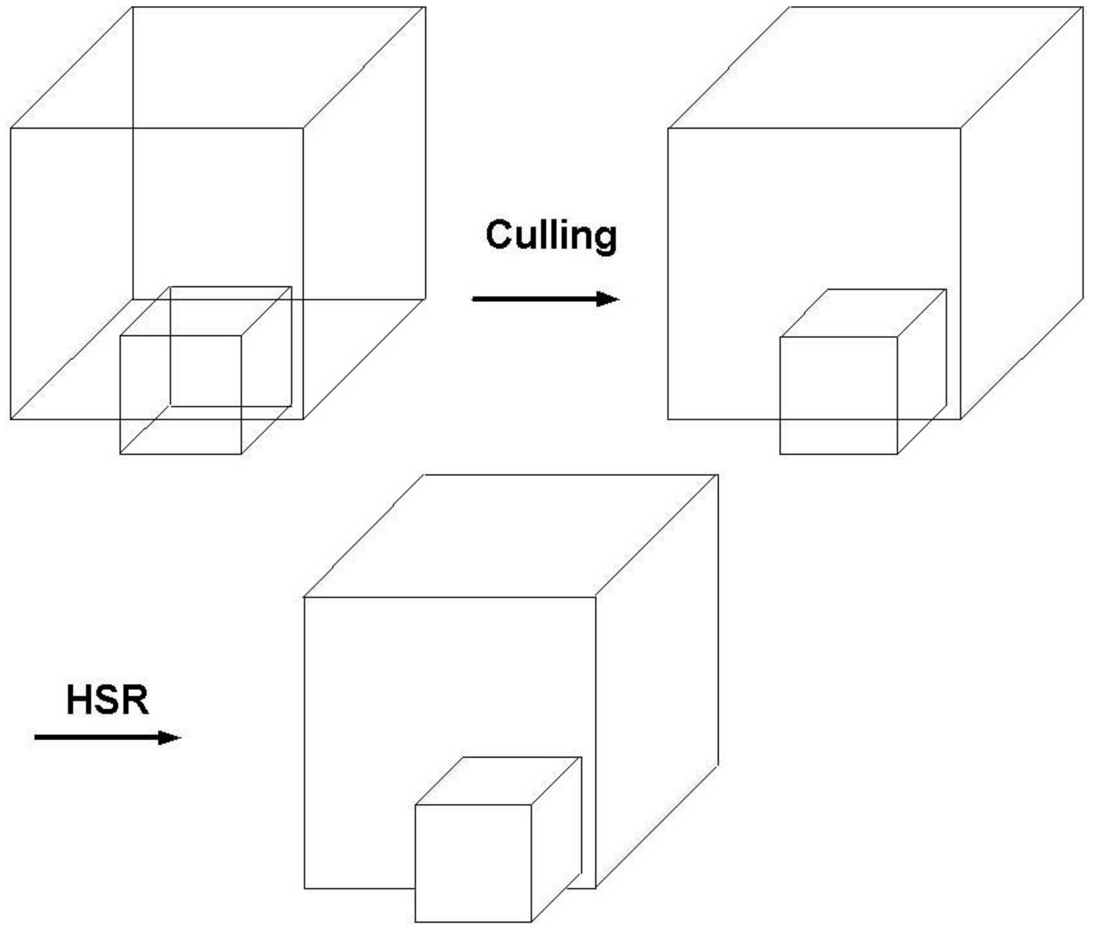
Steps – Rendering Pipeline

4. **Clipping:**
   – Truncate geometry outside the view volume
5. **Scan Conversion:**
   – Transform the polygonal model into a set of pixels (picture elements)
6. **Hidden Surface Removal (HSR):**
   – Remove invisible pixels, covered by other objects
7. **Shading:**
   – Find color (intensity) values of single pixels based on an illumination model

Remarks – Rendering Pipeline

- Step order can vary
- 5-7 are normally combined into a single step (Pixel Loop)
- Pipeline principle: Speed up
- 2 is a special case of 6: Performance!

Culling versus Hidden Surface Removal

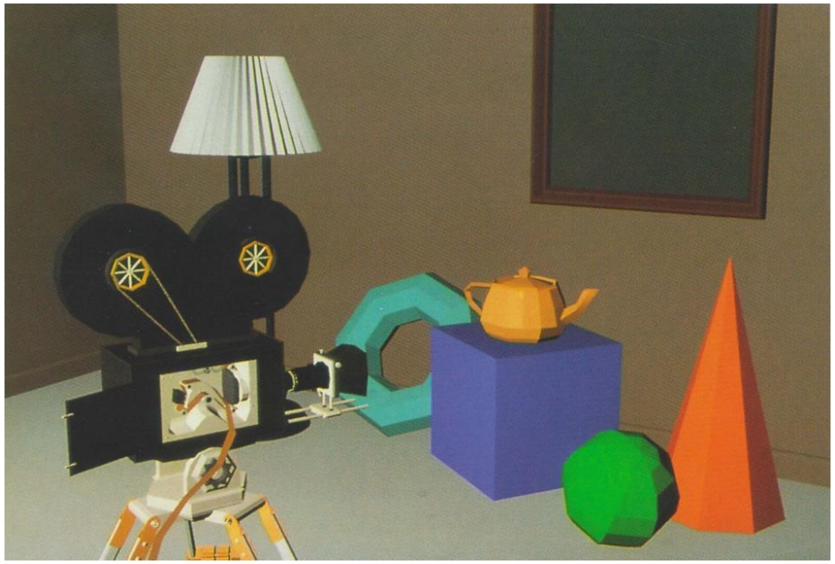


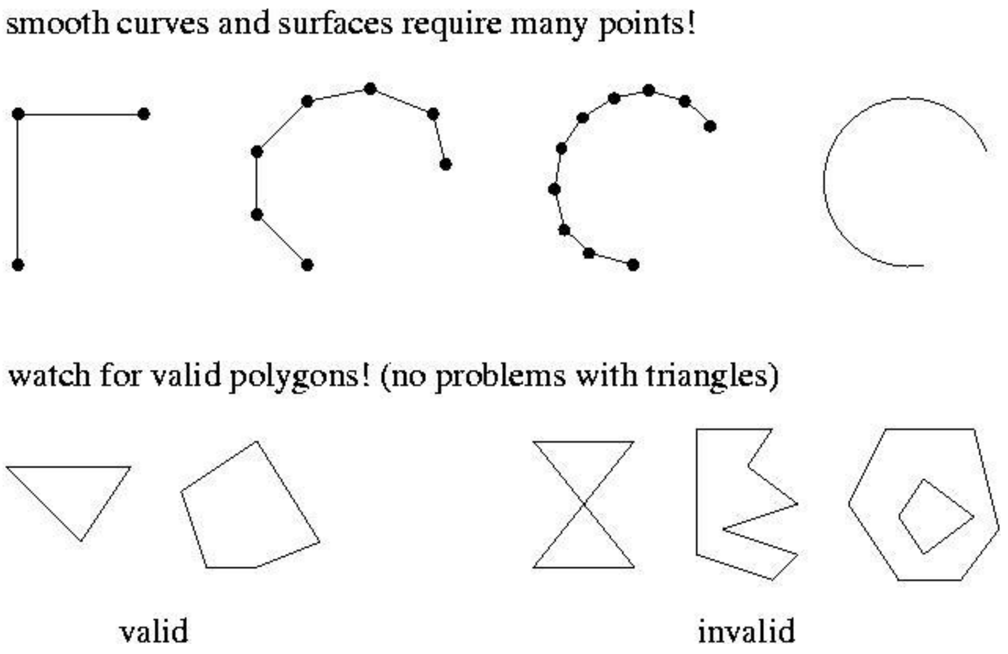
Topics – Basics in Computer Graphics / Rendering Pipeline

- **Rendering Pipeline**
- **Representation of rigid objects**
- **Transformations**
- **Culling**
- **Projection**
- **Clipping**
- **Hidden Surface Removal**
- **Shading**
- **Scan Conversion**
- **Texture Mapping**
- **Graphics Hardware**
- **Ray Tracing *(optional topic)***

Polygonal Representation – Object Representation

Approximation of object geometry by a grid of planar, polygonal facets



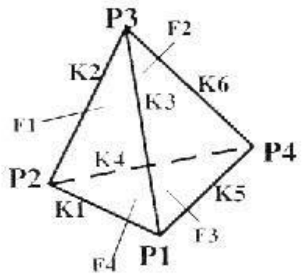

smooth curves and surfaces require many points!
watch for valid polygons! (no problems with triangles)
valid
invalid

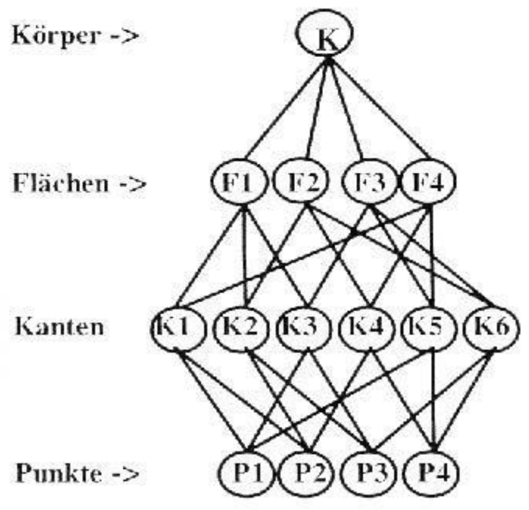
Pictures: Foley et al., Watt

Data Structure for Polygonal Representation – Object Representation

- List of 3-D coordinates (x, y, z) (polygon vertices)
- Edges implicitly
- Additional information: normals of polygons or polygon vertices for shading, color, material, …
- Hierarchical structure of single geometries (scene graph)
- Polygons are rendered independently from each other

- Drawback: Redundant rendering of common edges
- Alternative: explicit storage of edges – each edge is a list of 4 parameters: 2 adjacent vertices and 2 polygons





Picture: Schuhmann

Polygonal Representation - Benefits

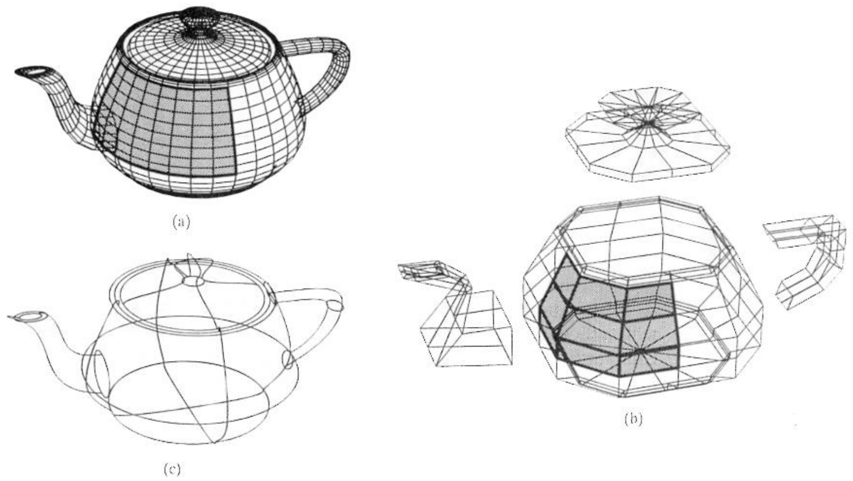
- Hardware Rendering
- Efficient Shading algorithms
- Arbitrary geometries
- Trade Off: Accuracy (number of polygons)

  versus

  speed (frame rate, polygons per second)

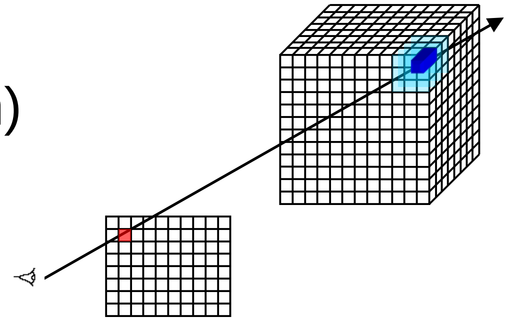
Alternatives – Object Representation

- Functional representation, e.g., sphere: $\boldsymbol{x^2 + y^2 + z^2 = r}$
- Constructive solid Geometry
- Bezier Patches



- Octrees
- Volume Rendering

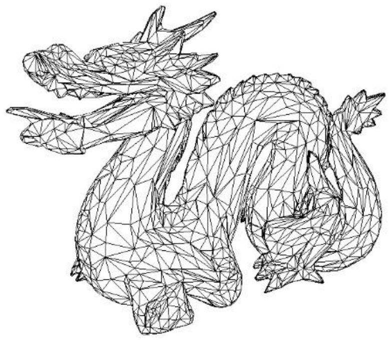
  (See lessons on visualization)



Topics – Basics in Computer Graphics / Rendering Pipeline

- Rendering Pipeline
- Representation of rigid objects
- Transformations
- Culling
- Projection
- Clipping
- Hidden Surface Removal
- Shading
- Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing *(optional topic)*

Overview – Rendering Pipeline



→ Transformation: Position und orientation of objects

→ Transformation: WKS, SKS

↓ Culling

← **Projection**

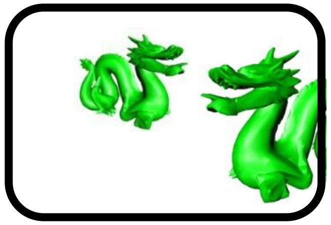
← Clipping

- -

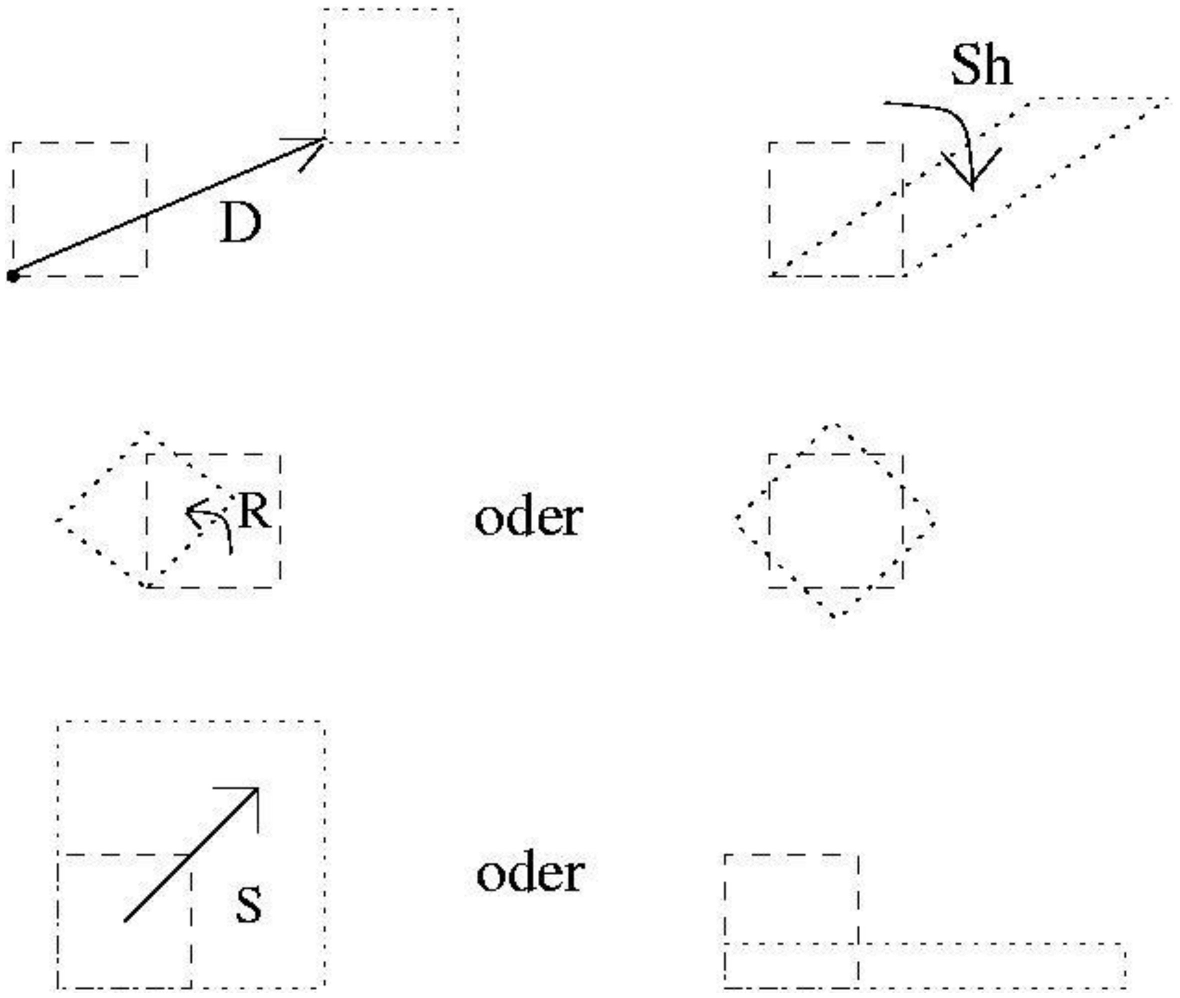
↓ Shading

→ Hidden Surface Removal

→ Scan Conversion

→ 

Basic Operations – Transformation



Basic Operations (II) - Transformation

Polygonal model: Just transform the vertices!

Conventions:

- Points are described as column or row vectors P (x, y, z)
- Transformations are described as matrices:
  - Translation $P' = P + D$ (D = translation vector)
  - Rotation $P' = P * R$ (R = rotation matrix)
  - Scaling $P' = P * S$ (S = scale matrix)
  - Shearing $P' = P * Sh$ (Sh = shear matrix)

Homogeneous Coordinates – Transformation

Inconsistency:

Translation is vector-vector-addition, all other operations are matrix vector-multiplication

Solution: Homogeneous coordinates – Increase dimension by 1

$P(x,y,z) \longrightarrow P(X,Y,Z,w)$, *with* $x=\frac{X}{w}, y=\frac{Y}{w}, z=\frac{Z}{w}$

Convention: $w=1$

Translation – Transformation

$$P' = P \cdot T$$

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

$$x' = x + T_x, \quad y' = y + T_y, \quad z' = z + T_z$$

Scaling – Transformation

$$P' = P \cdot S$$

$$S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x' = x \cdot S_x, \quad y' = y \cdot S_y, \quad z' = z \cdot S_z$$

Rotation – Translation

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about the z-axis:

$$x' = x\cos\theta - y\sin\theta \;,\; y' = y\sin\theta + y\cos\theta \;,\; z' = z$$

Application – Transformation

Concatenation of transformations by matrix-matrix-multiplication:

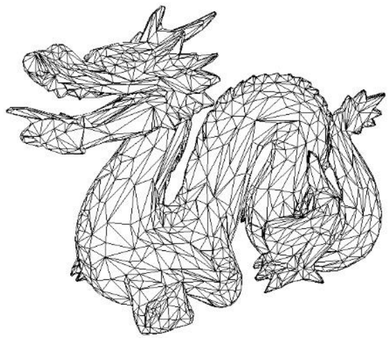
Representation of arbitrary linear transformations by a single matrix

- Operations on points of a single coordinate system
- Convert one coordinate system into another:
  - Local Object Coordinate Systems
  - Coordinate Systems for groups of objects
  - World Coordinate System
  - View Coordinate System
  - Screen Coordinate System

Topics – Basics in Computer Graphics / Rendering Pipeline

- Rendering Pipeline
- Representation of rigid objects
- Transformations
- Culling
- Projection
- Clipping
- Hidden Surface Removal
- Shading
- Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing *(optional topic)*

Overview – Rendering Pipeline



Transformation:
Position und orientation of objects

Transformation:
WKS, SKS

Culling

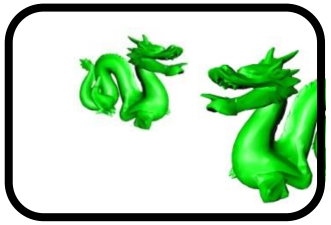
Projection

Clipping

Shading

Hidden Surface Removal

Scan Conversion

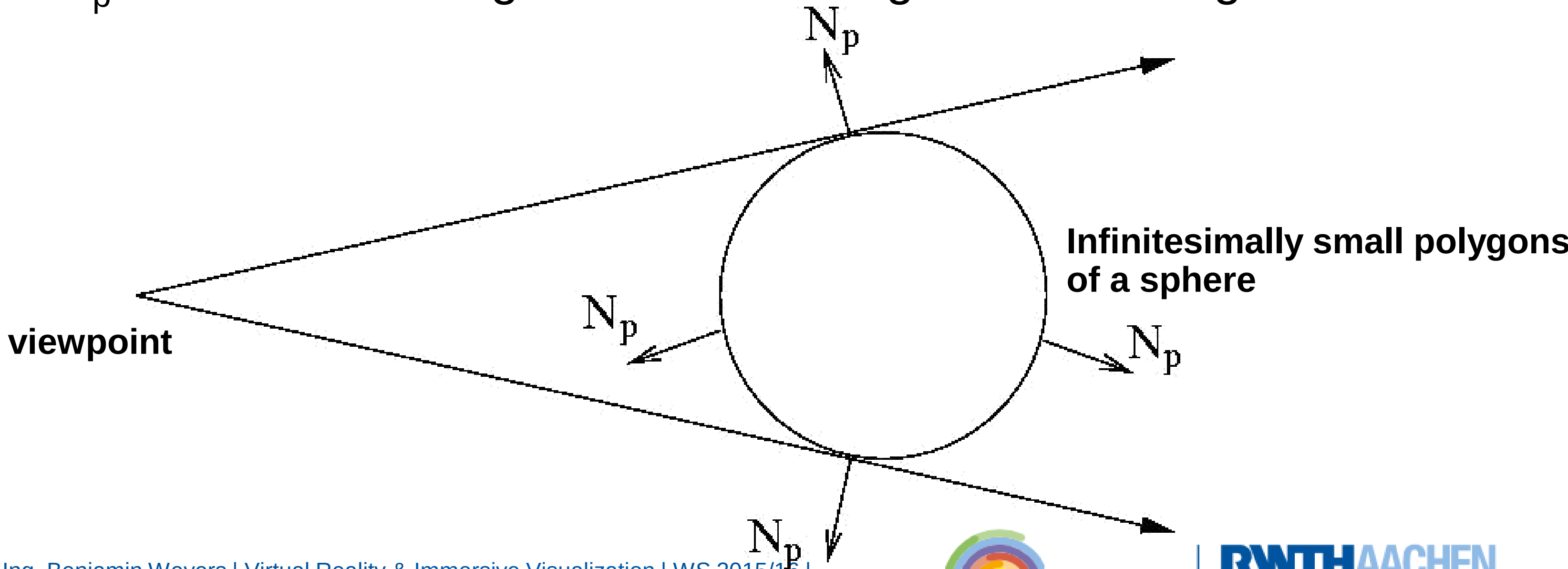


Algorithm – Culling

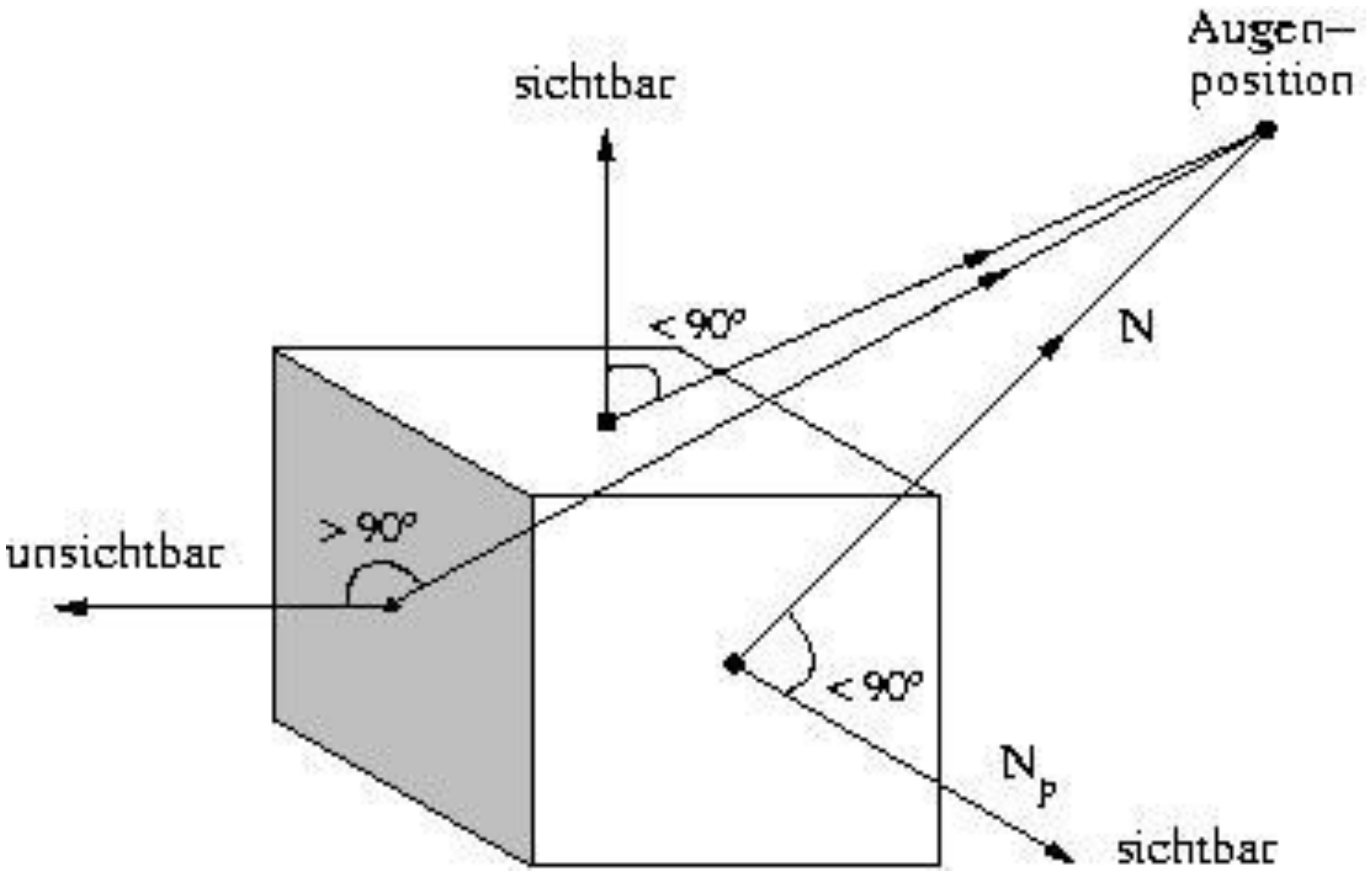
- Remove invisible polygons to speed up rendering (Culling, backface elimination)
- Compare polygon orientation with "line of sight" vector

Simple, fast „algorithm"

Polygon is invisible if and only if the angle between the polygon normal vector N_p and the line of sight vector N is larger than 90 degrees.



Backface Elimination of a Cube - Culling



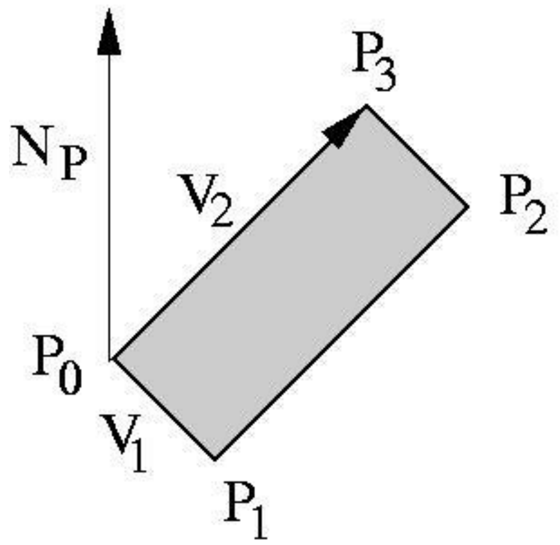
Picture based on Watt

Computation – Culling

$$\theta\left(N_P, N\right) < 90 \Leftrightarrow N_p \cdot N > 0 \quad \left(\cos\theta := \frac{N_p N}{\left|N_P\right|\left|N\right|}\right)$$

Determination of *N* and N_P:

- *N* is the position vector in the viewing coordinate system
- *Calculate* N_P from 3 (non-collinear) polygon vertices



$$V_1 = P_1 - P_0$$
$$V_2 = \mathbf{P_3} - P_0$$
$$N_P = V_1 \times V_2$$

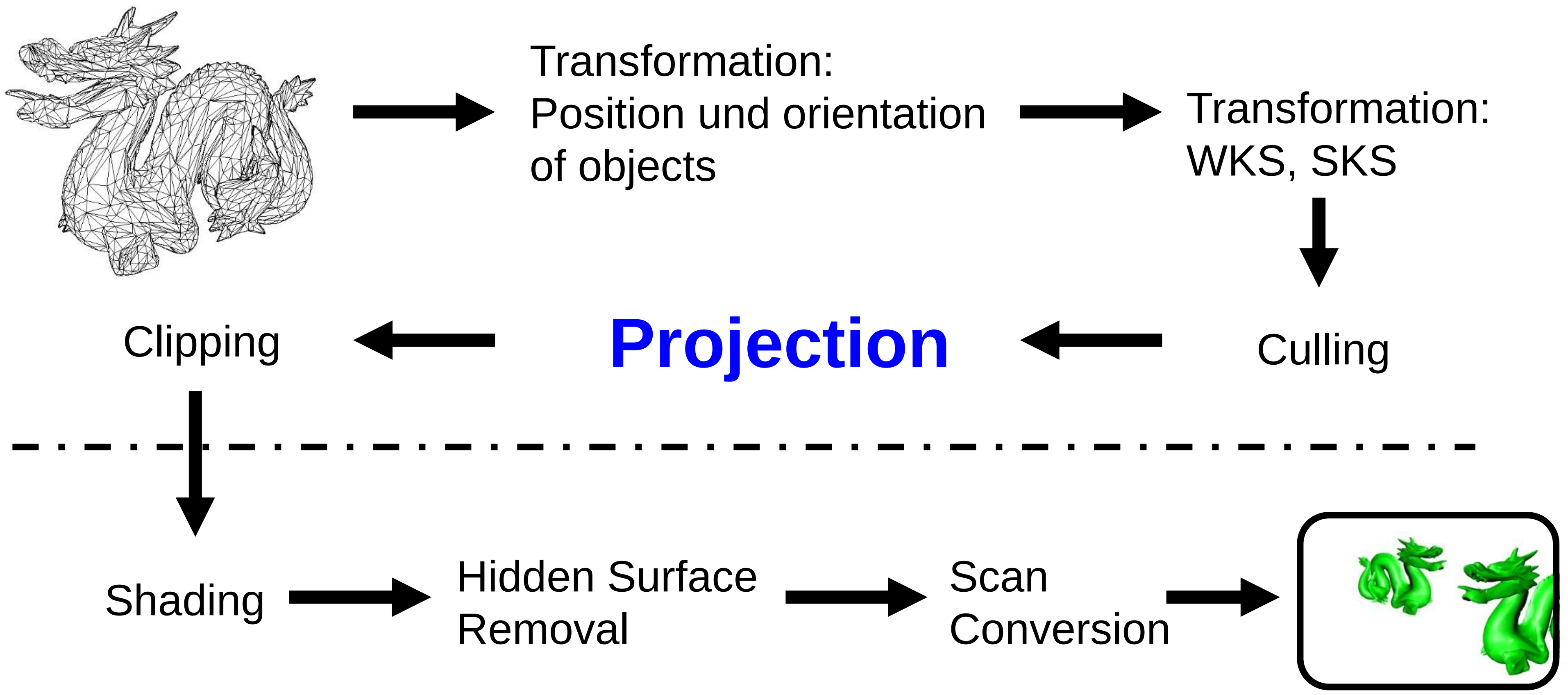
Culling – Remarks

- Compute N_Ps offline before the simulation starts, or store them with the model
- Reuse the normal vectors for shading
- At an average, half of the polygons of a polyhedron are invisible: Culling eliminates them at the very beginning of the rendering process

Topics – Basics in Computer Graphics / Rendering Pipeline

- **Rendering Pipeline**
- **Representation of rigid objects**
- **Transformations**
- **Culling**
- **Projection**
- **Clipping**
- **Hidden Surface Removal**
- **Shading**
- **Scan Conversion**
- **Texture Mapping**
- **Graphics Hardware**
- **Ray Tracing *(optional topic)***

Overview – Rendering Pipeline



Definition – Projection

- General:

  A projection is a function from n-dimensional space to (n-1)-dimensional space

- Computer Graphics:
  - 3-D $\Rightarrow$ 2-D (screen, projection plane, viewing window)
  - Projection plane is flat: straight lines are mapped to straight lines

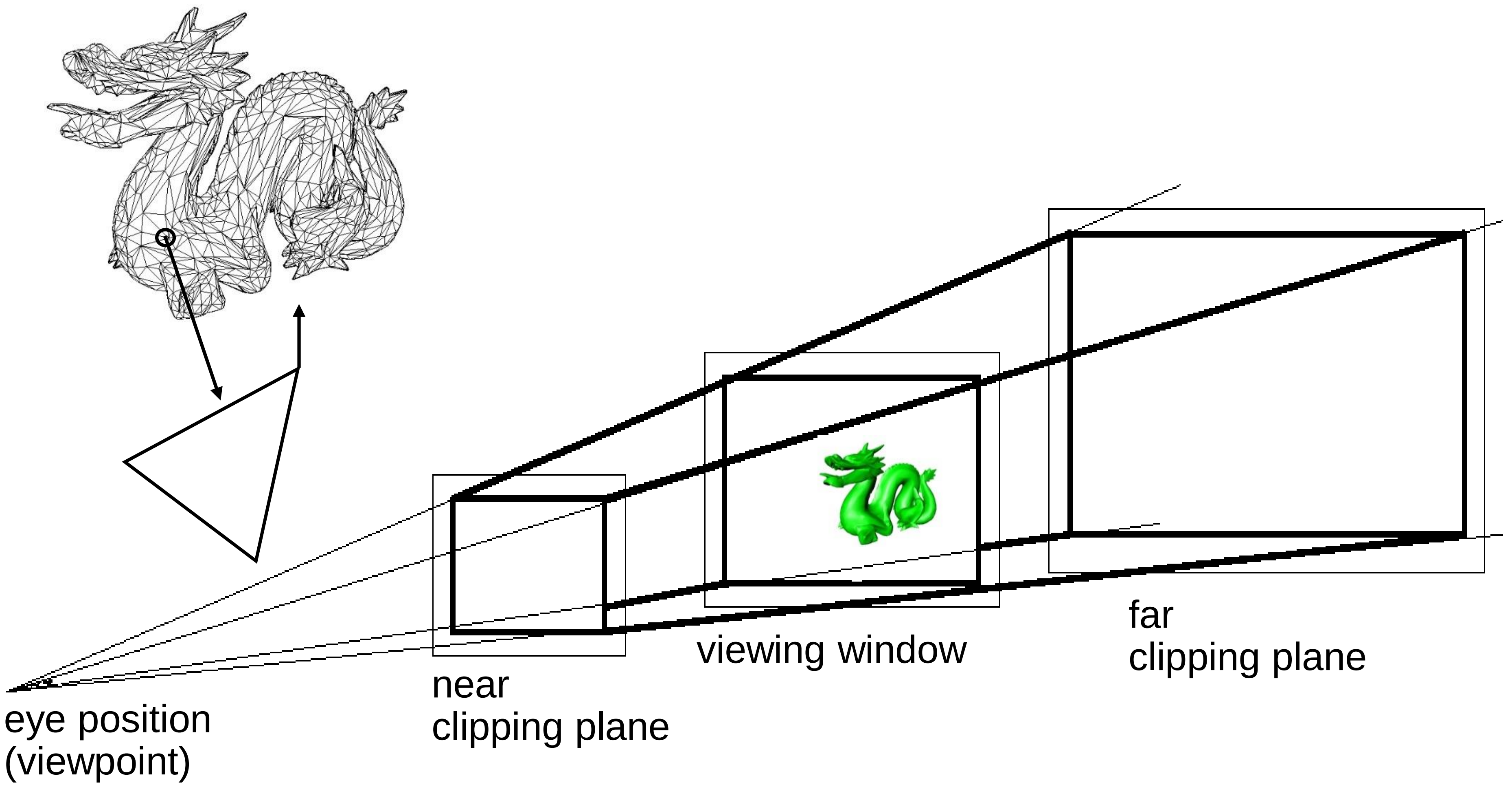
Perspective versus Parallel Projection - Projection

Perspective projection

- Projectors (projection beams) meet in the center of projection (eye position, viewpoint)
- Objects get „smaller“ with rising distance from the center of projection
- view volume is a pyramid

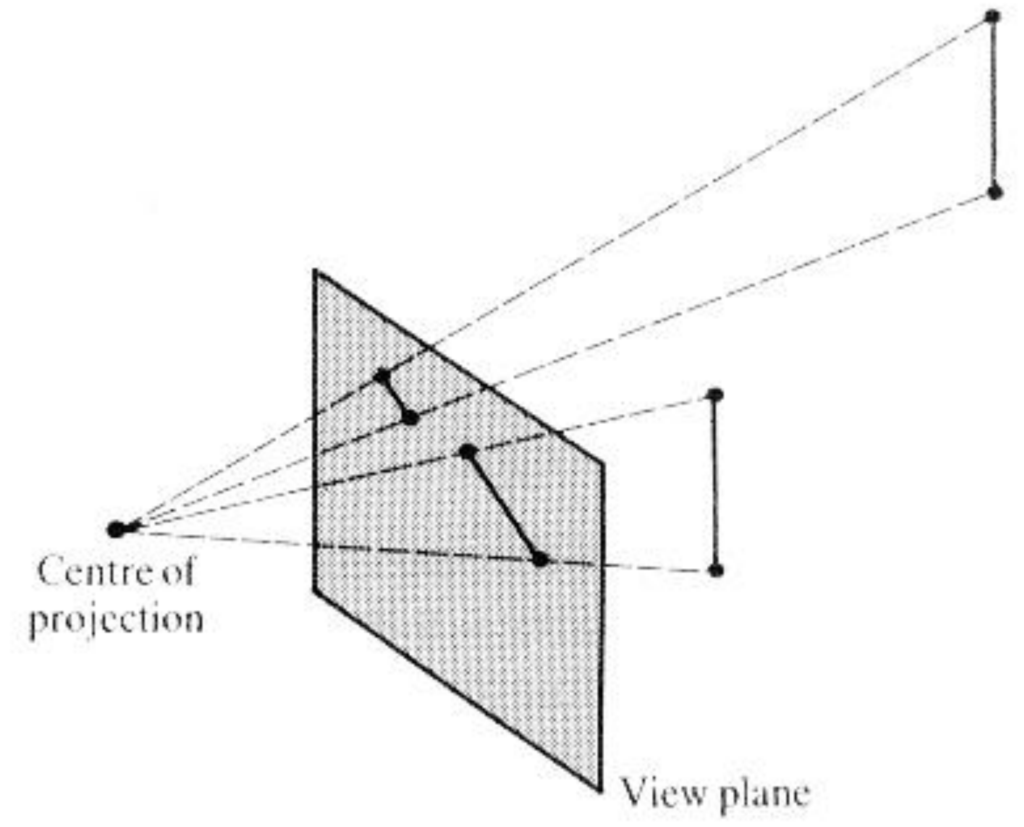
- **Parallel projection**
- Projectors (projection beams) are parallel, center of projection is in infinitum
- Does not correspond to natural viewing experience
- view volume is a cube

View Volume – Projection



Perspective Shortening – Projection

Picture: Watt

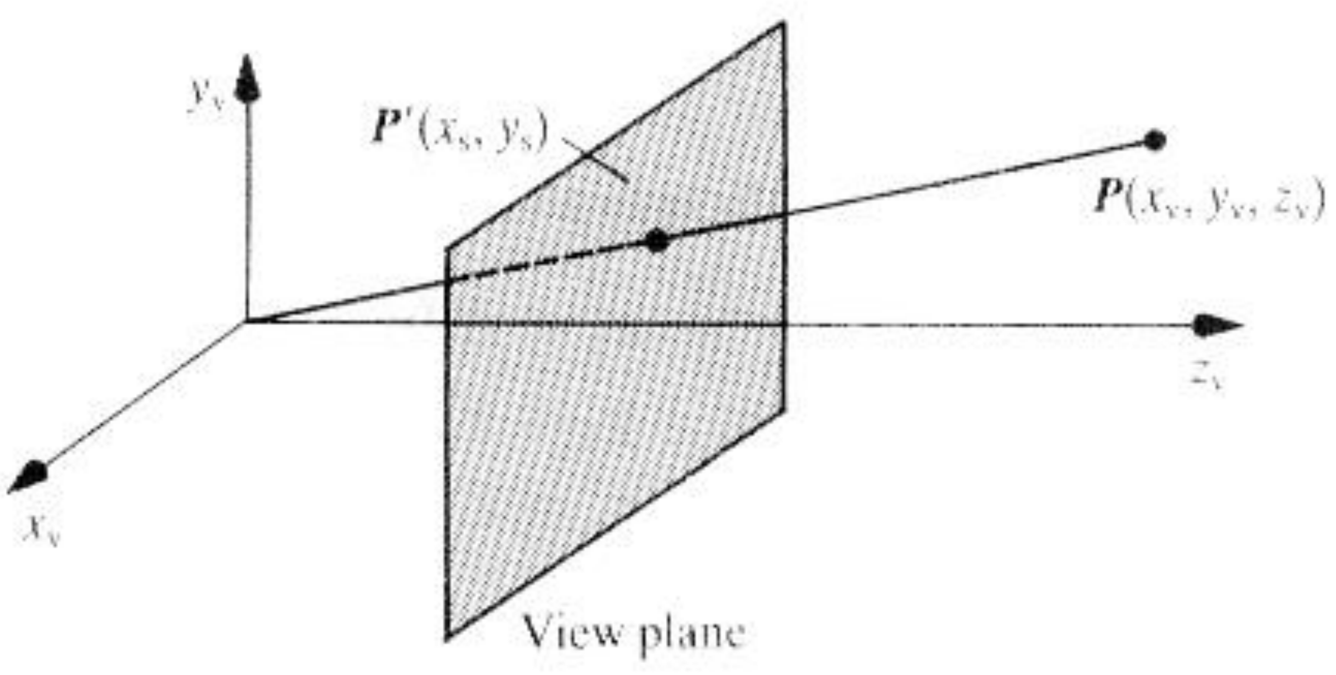


A very Simple Projection - Assumptions

- Perspective projection
- No „diagonal“ projection: Straight line from the center of projection to the center of the screen is parallel to the normal vector of the viewing window
- Constant distance *d* from the center of projection to the screen
- Computation:
  - View coordinate system
  - Normal vector of the viewing window is parallel to the z-axis

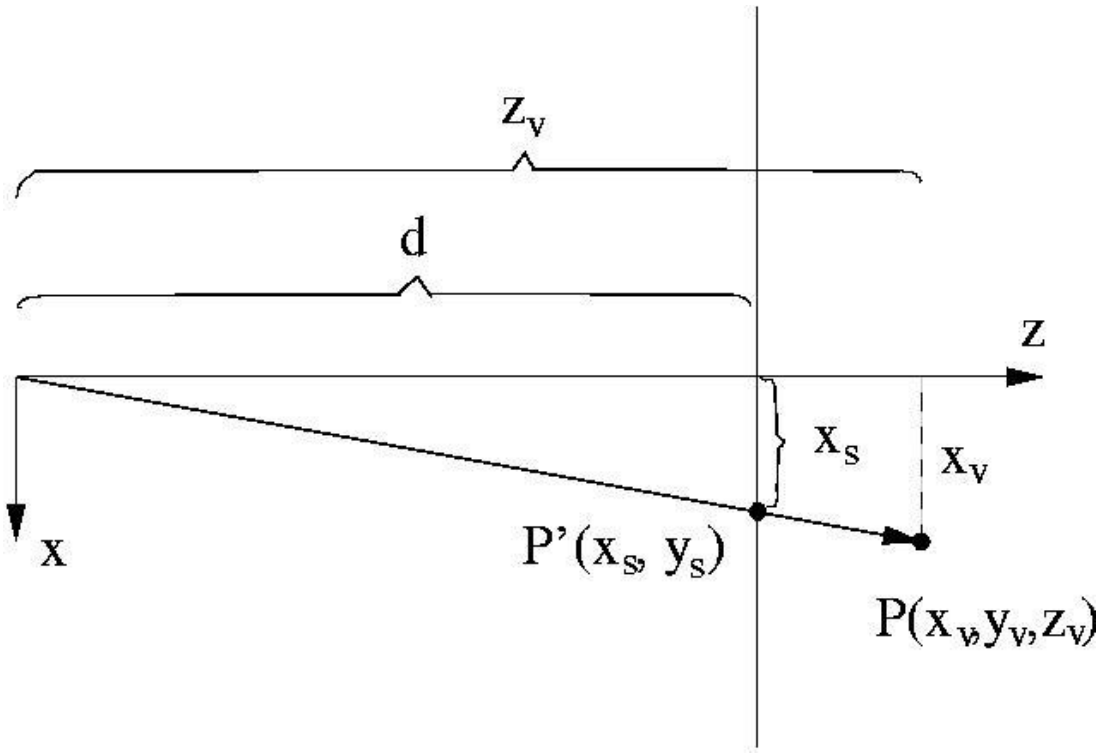
A very Simple Projection – Intersection Point

For each point $P(x_v, y_v, z_v)$ of the object geometry in 3-D space, we are looking for the intersection point $P'(x_s, y_s)$ of the projector (straight line from viewpoint to P) with the view plane.



A very Simple Projection – Computation

- Computation via similar triangles



$$\frac{x_s}{d} = \frac{x_v}{z_v} \quad (y_s \text{ analog}) \Rightarrow x_s = \frac{x_v d}{z_v} = \frac{x_v}{z_v / d} \quad \left(y_s = \frac{y_v d}{z_v} = \frac{y_v}{z_v / d}\right)$$

A very Simple Projection – Projection Matrix

Homogeneous coordinates:

$$X = x_V\,,\; Y = y_V \quad , \quad Z = z_V \quad , \quad w = \frac{z_V}{d}$$

$$(X\;Y\;Z\;w) = \left(x_V\;\; y_V\;\; z_V\;\; 1\right) T_{Proj} \quad mit$$

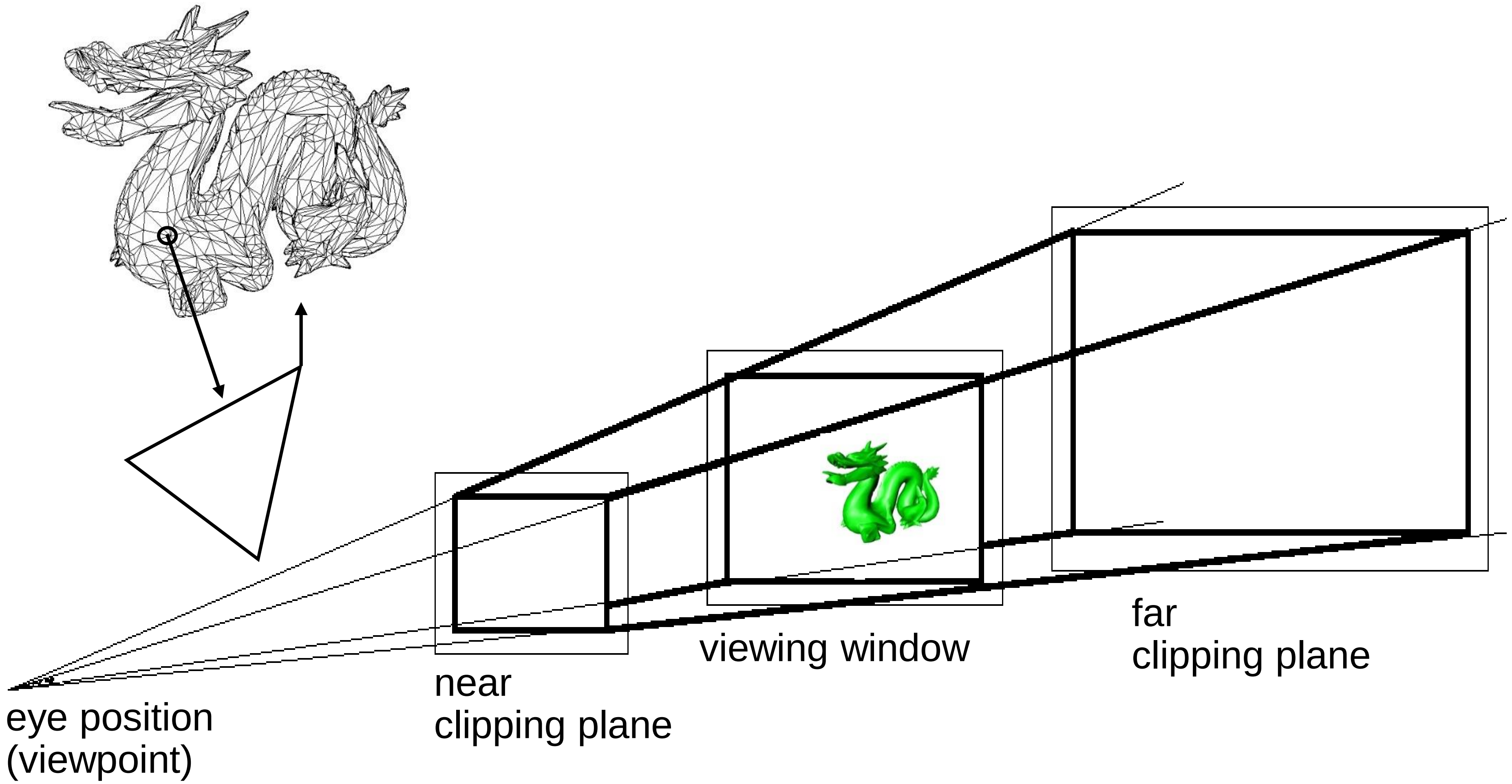
$$T_{Proj} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{d} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$x_s = \frac{X}{w}, y_s = \frac{Y}{w}, z_s = \frac{Z}{w} = d$$

Topics – Basics in Computer Graphics / Rendering Pipeline

- Rendering Pipeline
- Representation of rigid objects
- Transformations
- Culling
- Projection
- Clipping – will be skipped in this course
- Hidden Surface Removal
- Shading
- Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing *(optional topic)*

View Volume – Clipping



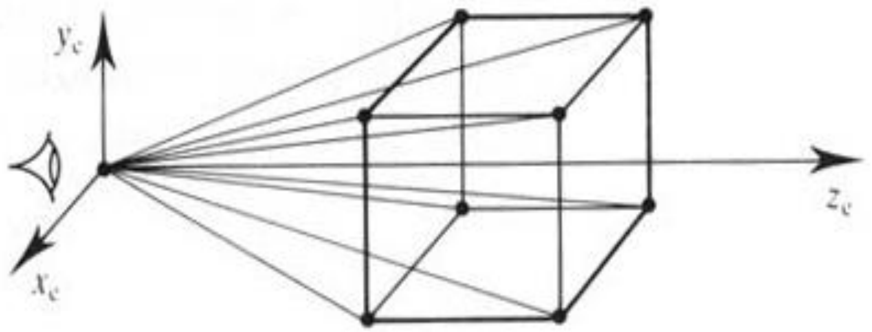
Topics – Basics in Computer Graphics / Rendering Pipeline

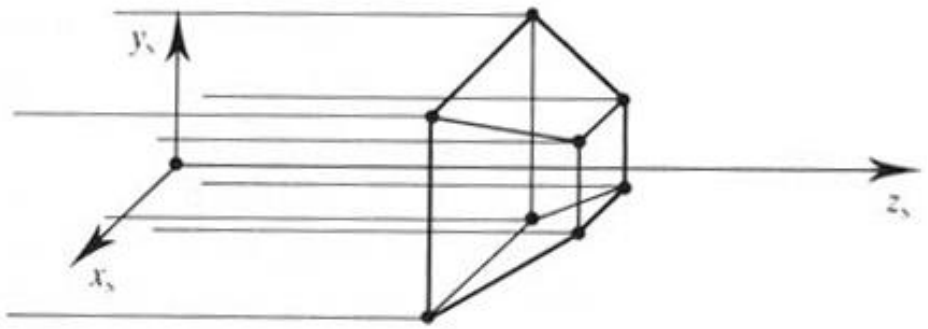
- **Rendering Pipeline**
- **Representation of rigid objects**
- **Transformations**
- **Culling**
- **Projection**
- **Clipping**
- **Hidden Surface Removal**
- **Shading**
- **Scan Conversion**
- **Texture Mapping**
- **Graphics Hardware**
- **Ray Tracing *(optional topic)***

Z-Buffer – Hidden Surface Removal

Z-Buffer

- Special memory on graphics hardware, 1 entry for every pixel
- Updated with the highest z value of 3-D object points that cover its pixel
- Accuracy depends on
  - Length of memory words („depth“) (usually 24 bits/pixel)
  - Z-value of front and back clipping plane
- Compare points lying on the same projector (OpenGL: parallel projection!)

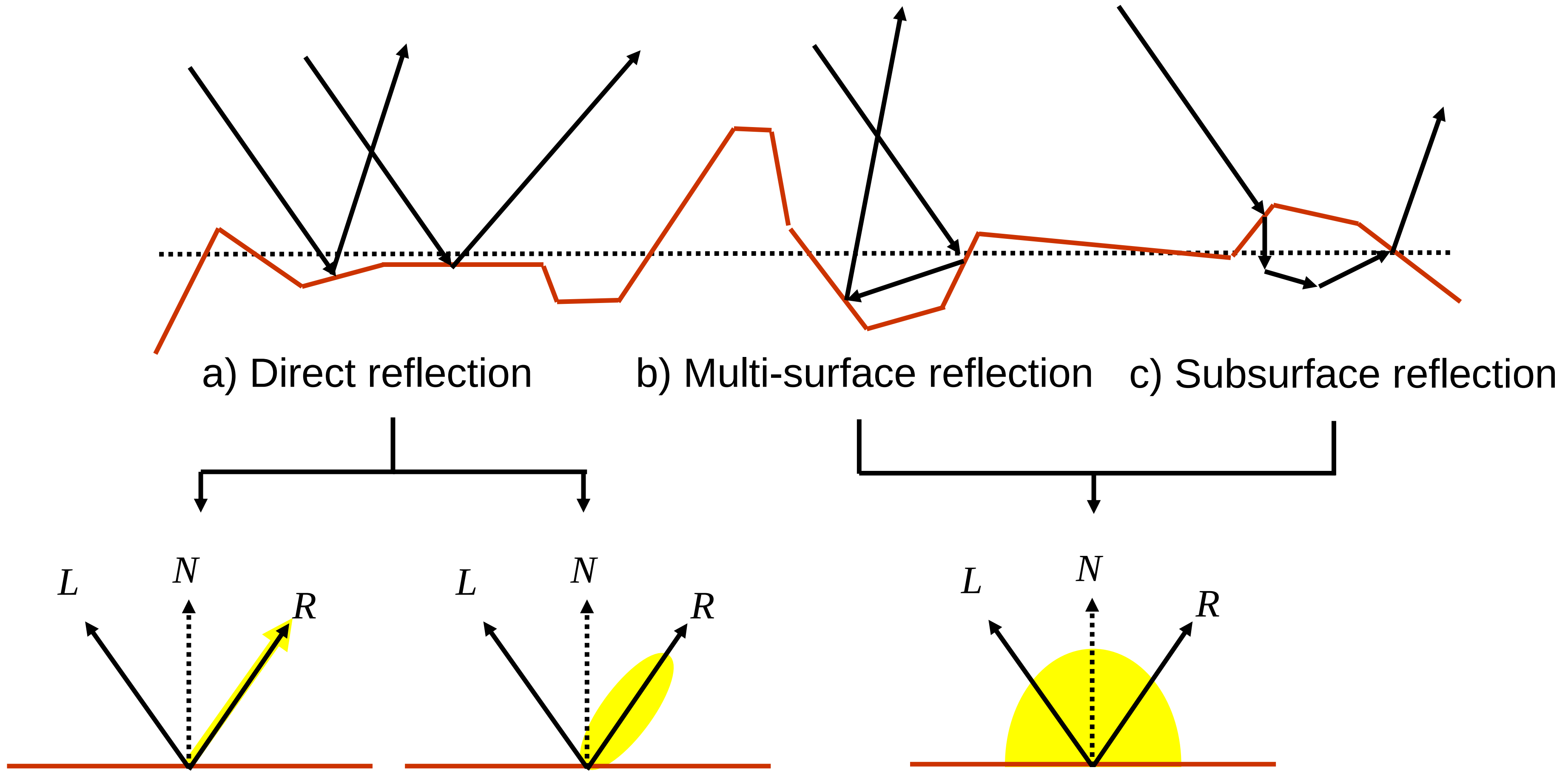




Topics – Basics in Computer Graphics / Rendering Pipeline

- Rendering Pipeline
- Representation of rigid objects
- Transformations
- Culling
- Projection
- Clipping
- Hidden Surface Removal
- Shading
- Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing *(optional topic)*

Shading – Reflection on Surfaces



1) Mirror reflection (perfect specular reflection)

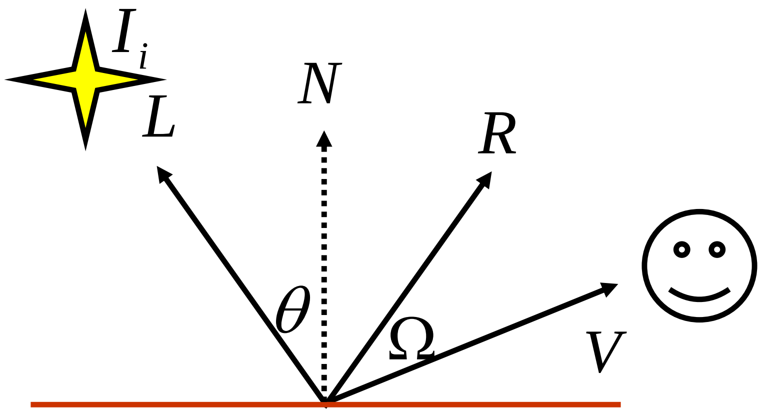
2) Directed diffuse reflection

3) Diffuse Reflection

Shading – Phong Reflection Model

Linear combination from 3 components:

- Diffuse
- Specular
- Ambient



Reflection coefficients: k_d, k_s, k_a

Diffuse component: $I_d = I_i k_d \cos\theta = I_i k_d (L \cdot N)$ $\quad I_d = k_d \sum_n I_{i,n} (L_n \cdot N)$

Specular component: $I_s = I_i k_s \cos^n \Omega = I_i k_s (R \cdot V)^n$

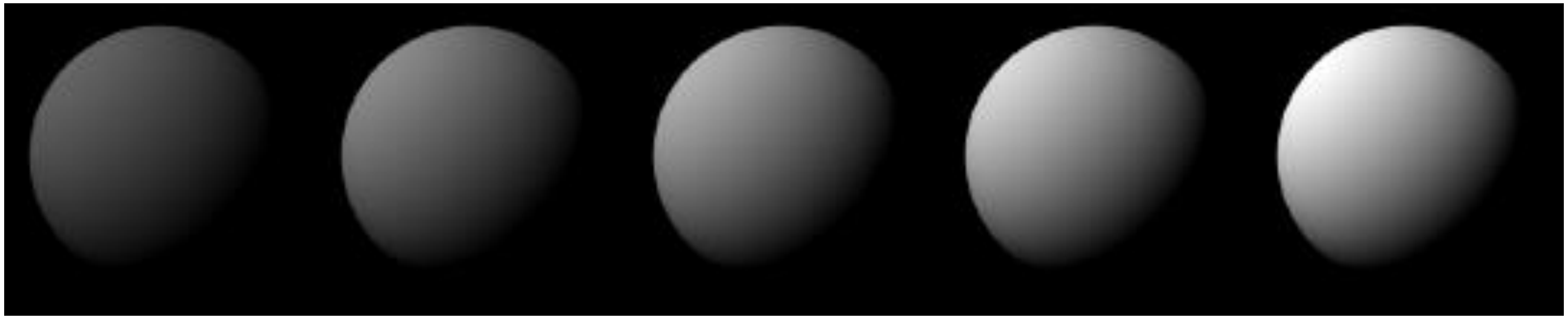
n: Index for surface roughness

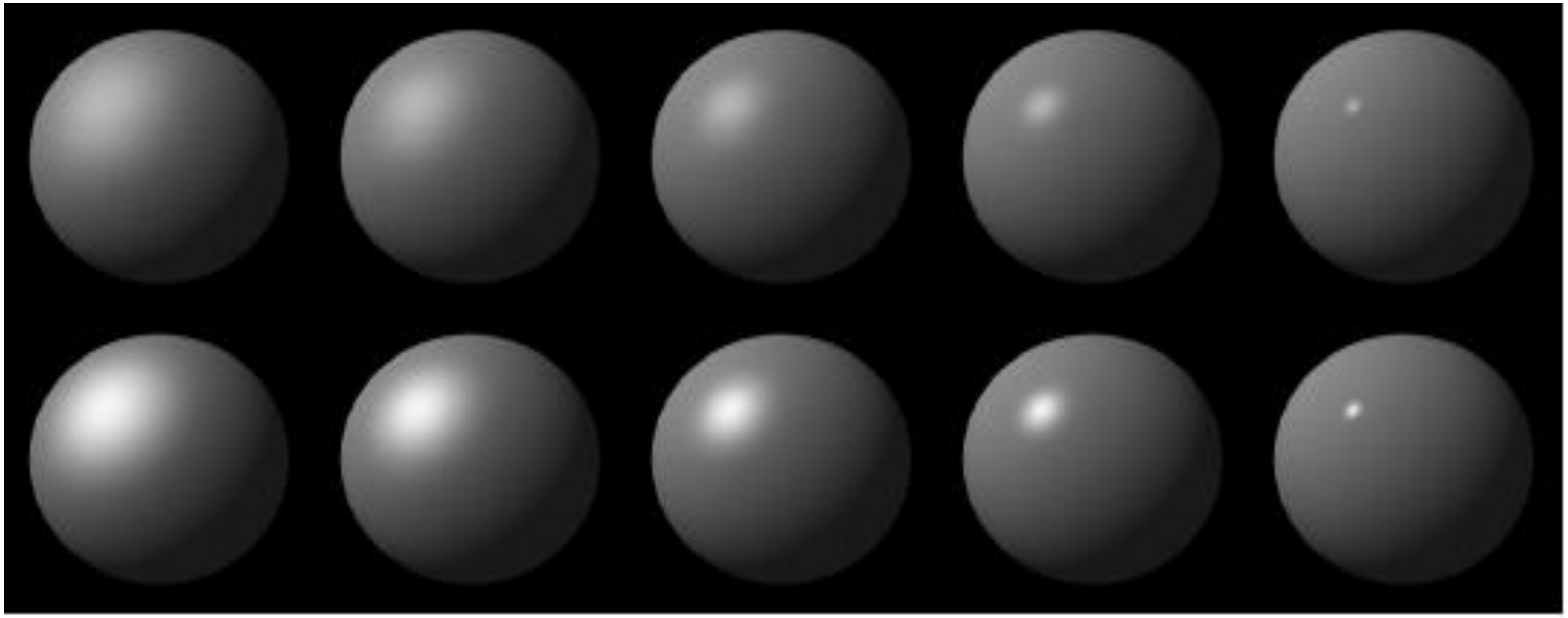
Perfect mirror: $n \rightarrow \infty$ (Ray Tracing: Recursion)

Ambient component: $I_g = I_a k_a$

Overall intensity: $I = I_a k_a + I_i (k_d (L \cdot N) + k_s (R \cdot V)^n)$

Shading – Diffuse vs. Specular

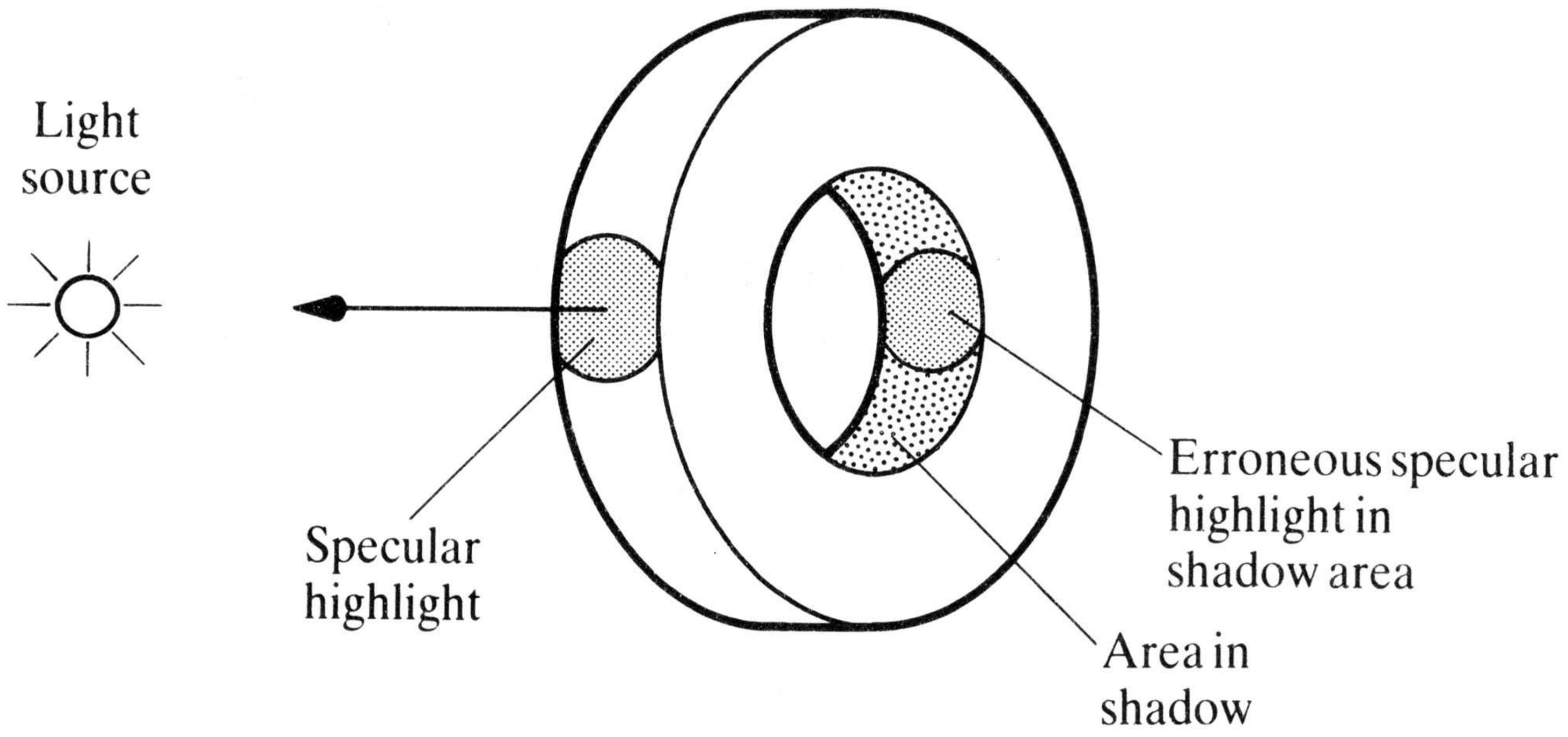




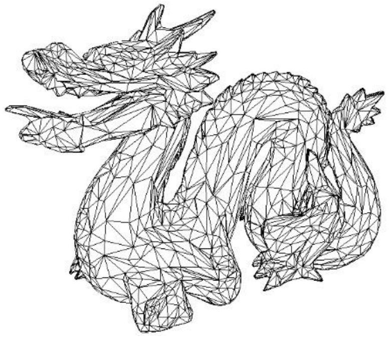
Pictures: Westermann

Shading – Phong Reflection Model - Errors

Picture: Watt



Overview – Rendering Pipeline



Transformation:
Position und orientation
of objects

Transformation:
WKS, SKS

Culling

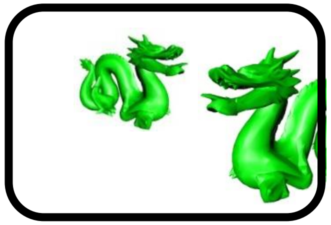
Projection

Clipping

Shading

Hidden Surface
Removal

Scan
Conversion



Shading – What is Shading?

Shading

Definition: (Incremental, interpolative)

Application of a reflection model on polygons by calculation of intensities at polygon vertices and interpolation of these values for the inner points

CG: Phong reflection model

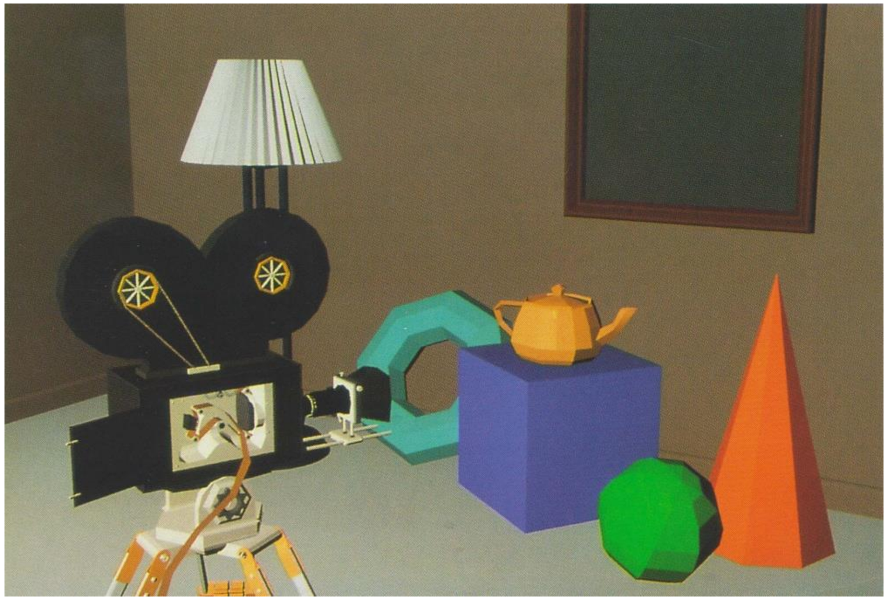
1. Flat Shading
2. Gouraud Shading
3. Phong Shading

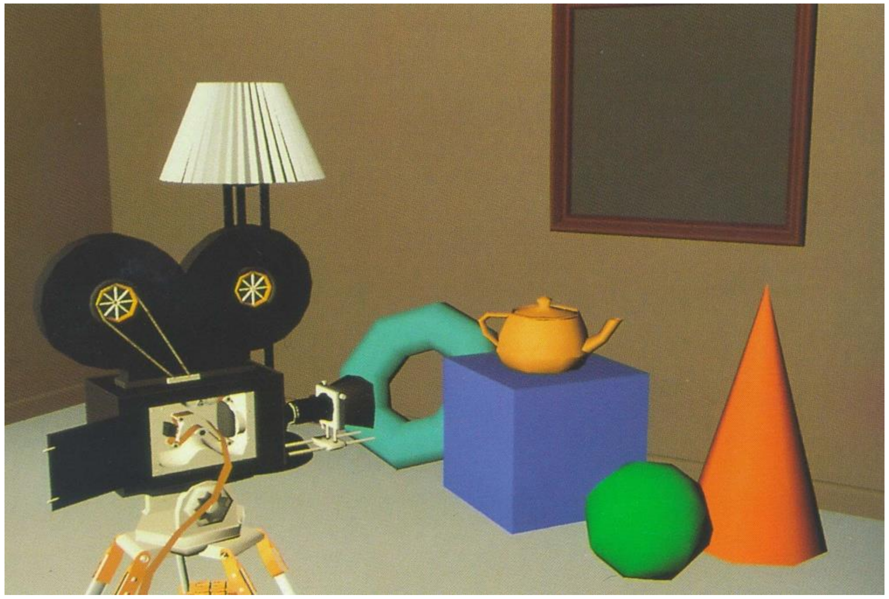
Shading – Flat Shading vs. Gouraud Shading

- **Flat Shading**

  No interpolation, all inner points of a polygon get the same intensity
- **Gouraud Shading**

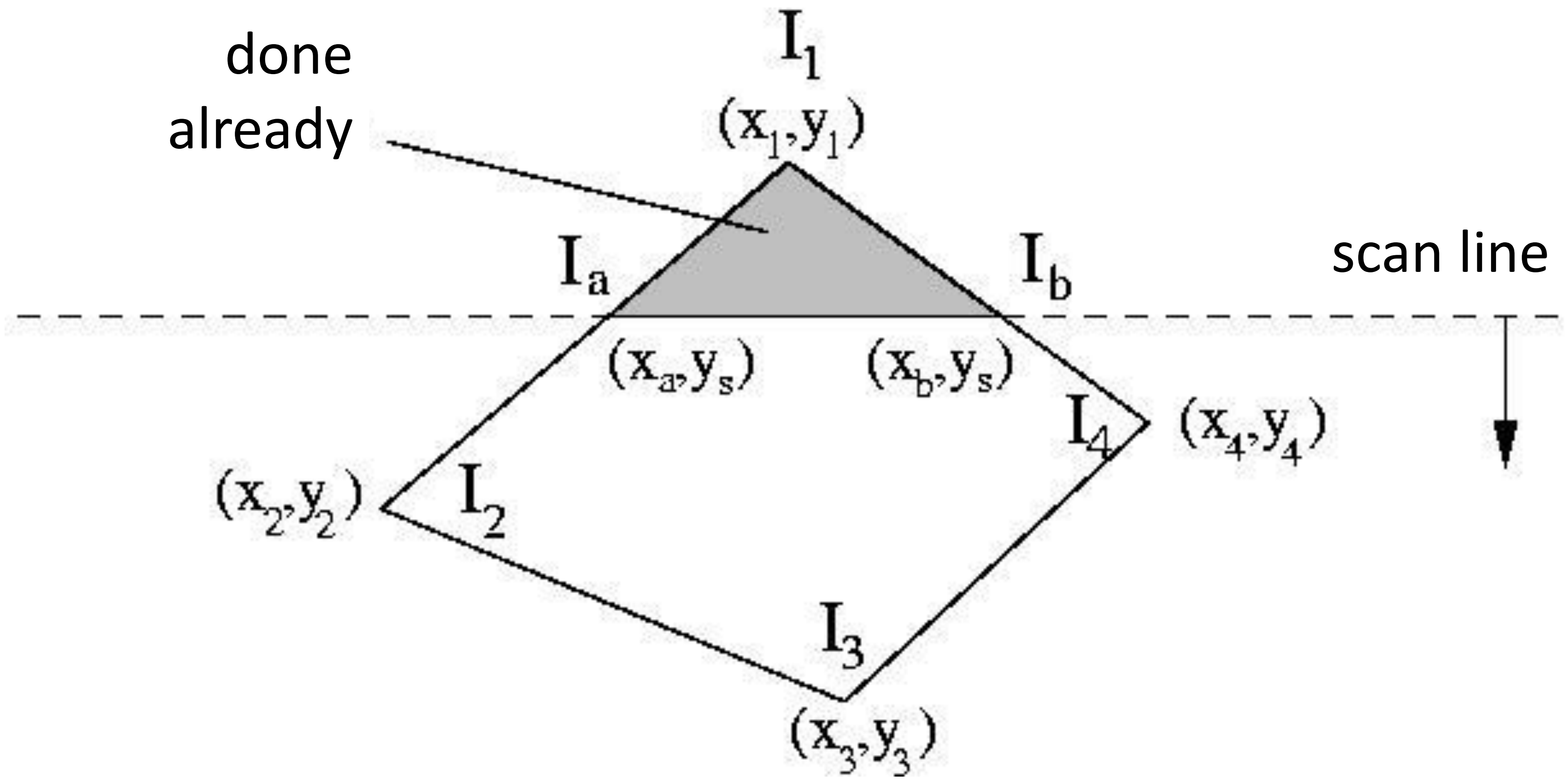
  (Bilinear) interpolation of the inner points from the vertex intensities





Pictures: Foley et al.

Gouraud Shading – Bilinear Interpolation



Gouraud Shading – Diffuse Component

- Calculate the vertex normal vector as average from the adjacent polygon normal vectors (offline!)
- Calculate the vertex intensities according to Phong model
- Interpolation process (integrated in scan conversion) in scan line order
  - Interpolate intensities at the intersection points of scan line and polygon edges from the vertex intensities

$$I_a = \frac{1}{y_1 - y_2}\left(I_1\left(y_s - y_2\right) + I_2\left(y_1 - y_s\right)\right)$$

$$I_b = \frac{1}{y_1 - y_4}\left(I_1\left(y_s - y_4\right) + I_4\left(y_1 - y_s\right)\right)$$

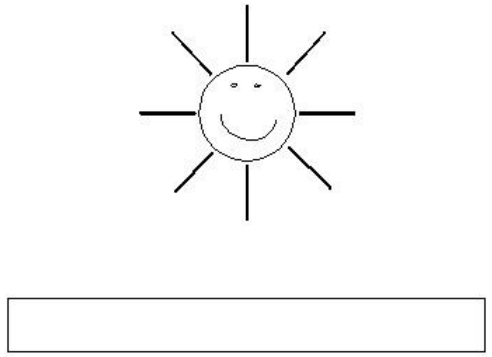
  - Interpolate intensities for the inner points along the scan line from the intensities at the intersection points

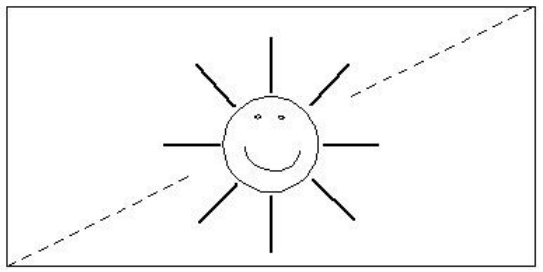
$$I_s = \frac{1}{x_b - x_a}\left(I_a\left(x_b - x_s\right) + I_b\left(x_s - x_a\right)\right)$$

Has to be calculated for every pixel: incremental approach, see literature

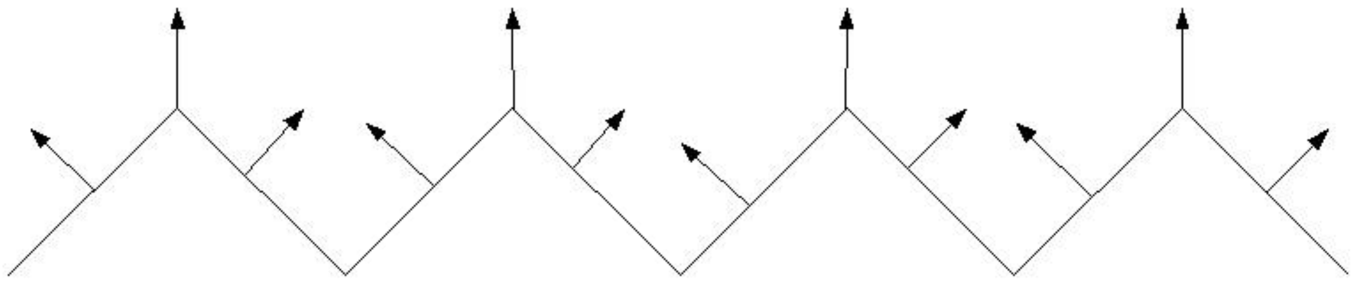
Gouraud Shading – Some Problems

- No highlights in the middle of polygons (example: virtual table)



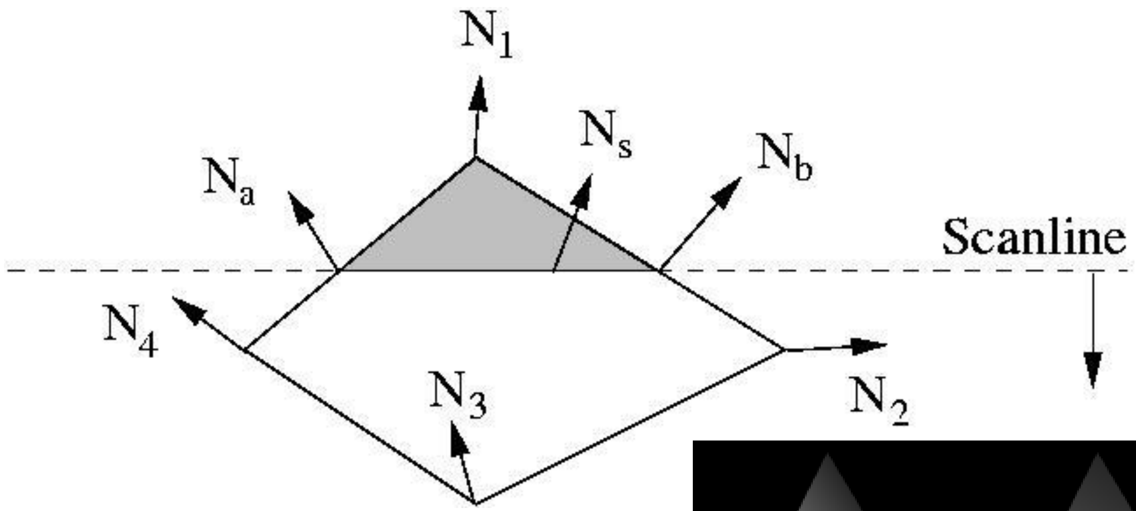


- Corrugations are smoothed out

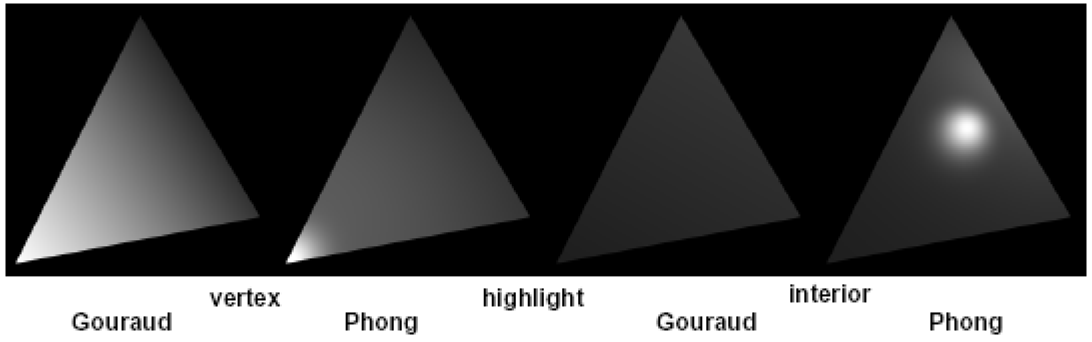


Phong Shading

- Interpolation of vertex normal vectors instead of intensities
- Get an individual normal vector for each pixel as an approximation of the "real" normal vector on a curved surface



Picture: Westermann



Phong Shading – Interpolation of Normal Vectors

$$N_a = \frac{1}{y_1 - y_2}\left(N_1(y_s - y_2) + N_2(y_1 - y_s)\right)$$

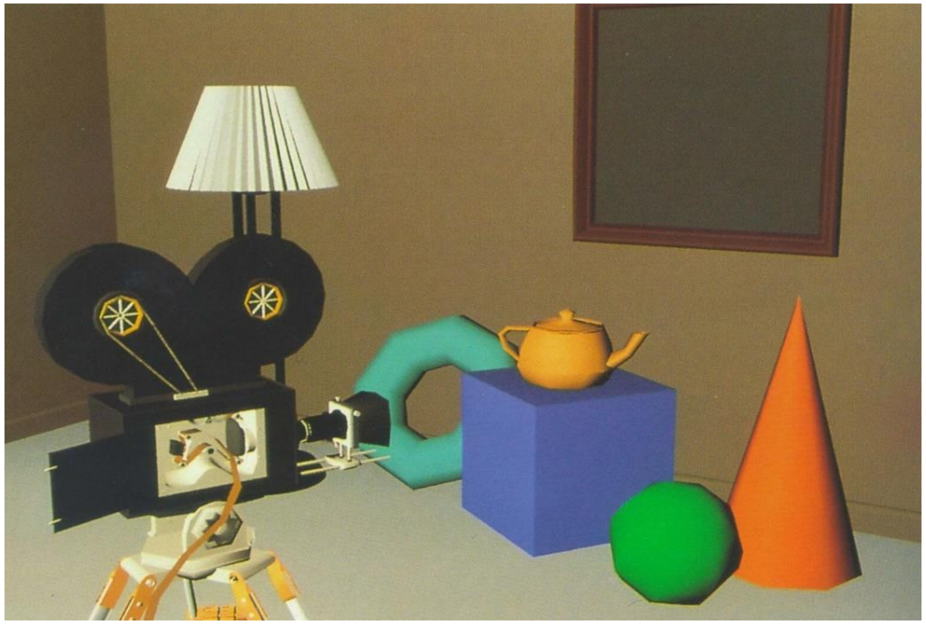
$$N_b = \frac{1}{y_1 - y_4}\left(N_1(y_s - y_4) + N_4(y_1 - y_s)\right)$$

$$N_s = \frac{1}{x_b - x_a}\left(N_a(x_b - x_s) + N_b(x_s - x_a)\right)$$

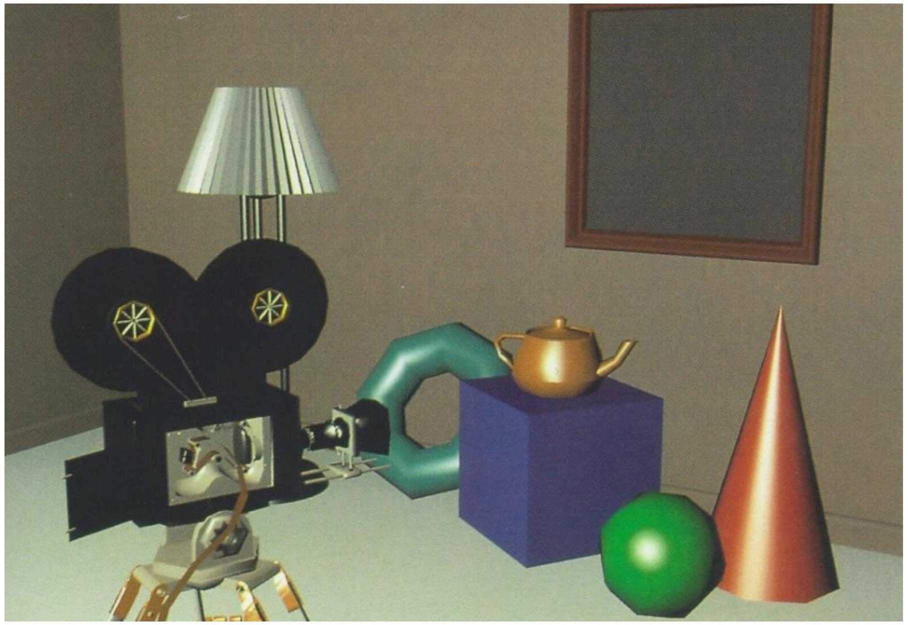
- Phong: three times more complex than Gouraud
- Also: Calculate intensity at each pixel
- Speed up: Combine Gouraud and Phong Shading (Interpolate normal vector for every second pixel and interpolate intensity for the other pixels)
- Phong shading never realized in graphics hardware (?) – can be simulated with textures

Gouraud Shading vs. Phong Shading

Pictures: Foley et al.



Simple Gouraud Shading

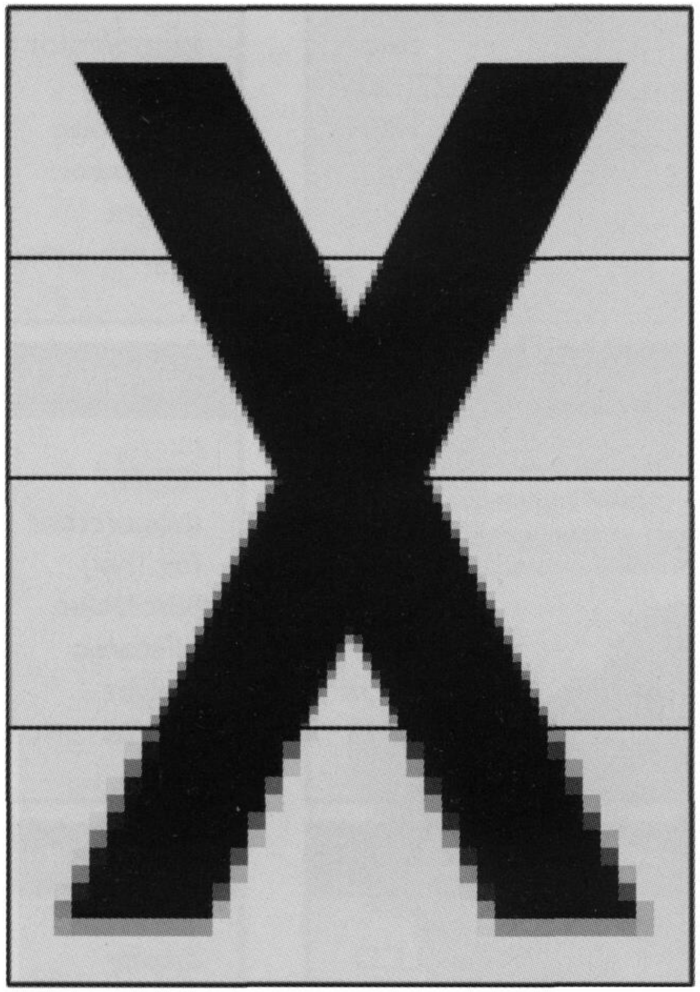


Phong Shading
with Specular Highlights

Topics – Basics in Computer Graphics / Rendering Pipeline

- Rendering Pipeline
- Representation of rigid objects
- Transformations
- Culling
- Projection
- Clipping
- Hidden Surface Removal
- Shading
- Scan Conversion will be skipped, but Supersampling!
- Texture Mapping
- Graphics Hardware
- Ray Tracing *(optional topic)*

Display Resolution



XGA 1999

VGA 1997

NTSC 1995

CGA 1993

Picture: Burdea et al.

Today:

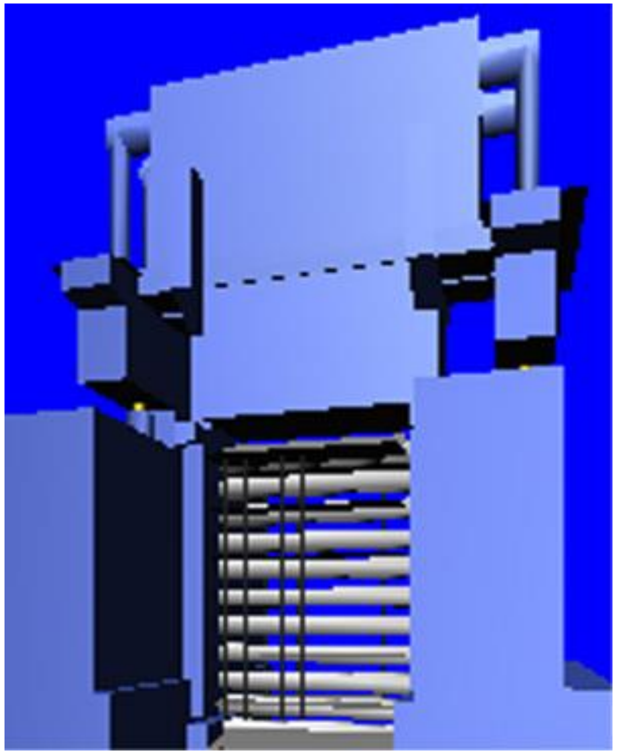
- SXGA, 1280x1024
- UXGA, 1600x1200
- WUXGA, 1920x1200

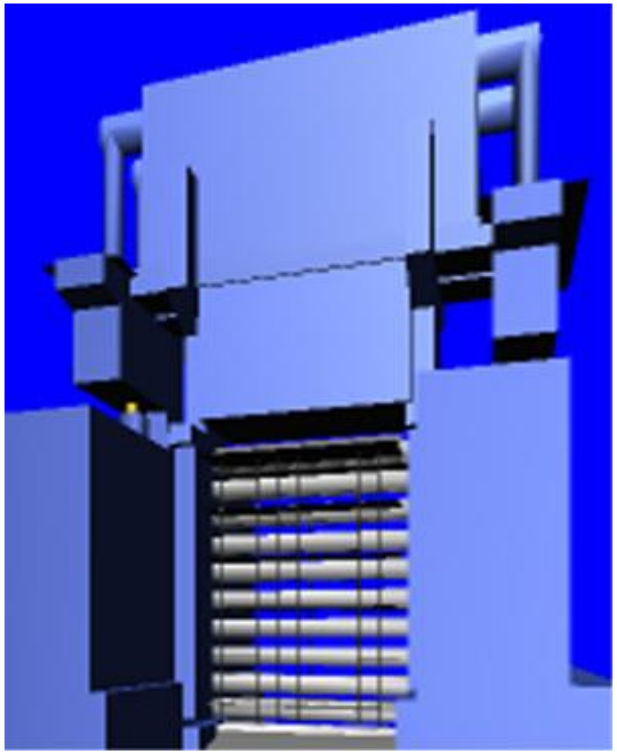
See also:

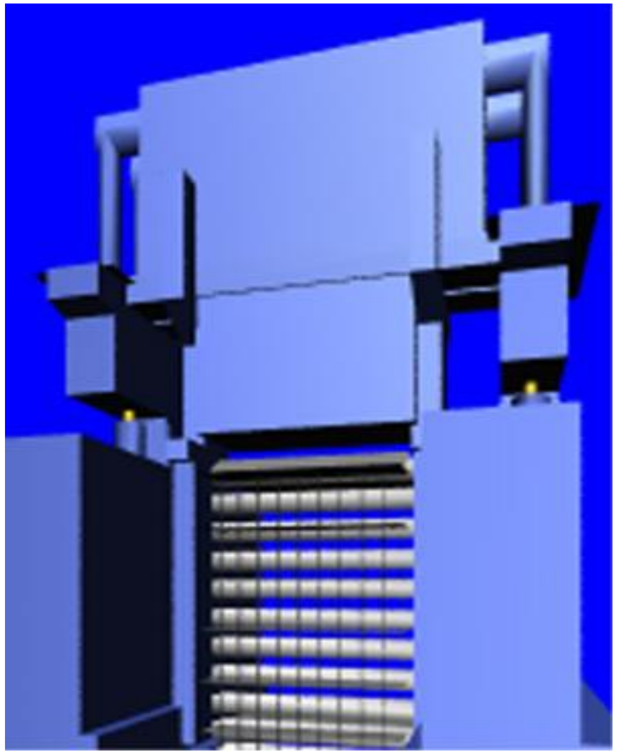
- Color depth
- Anti-Aliasing

Scan Conversion – Anti-Aliasing

Pictures: IT Center RWTH


1 sample


4 samples

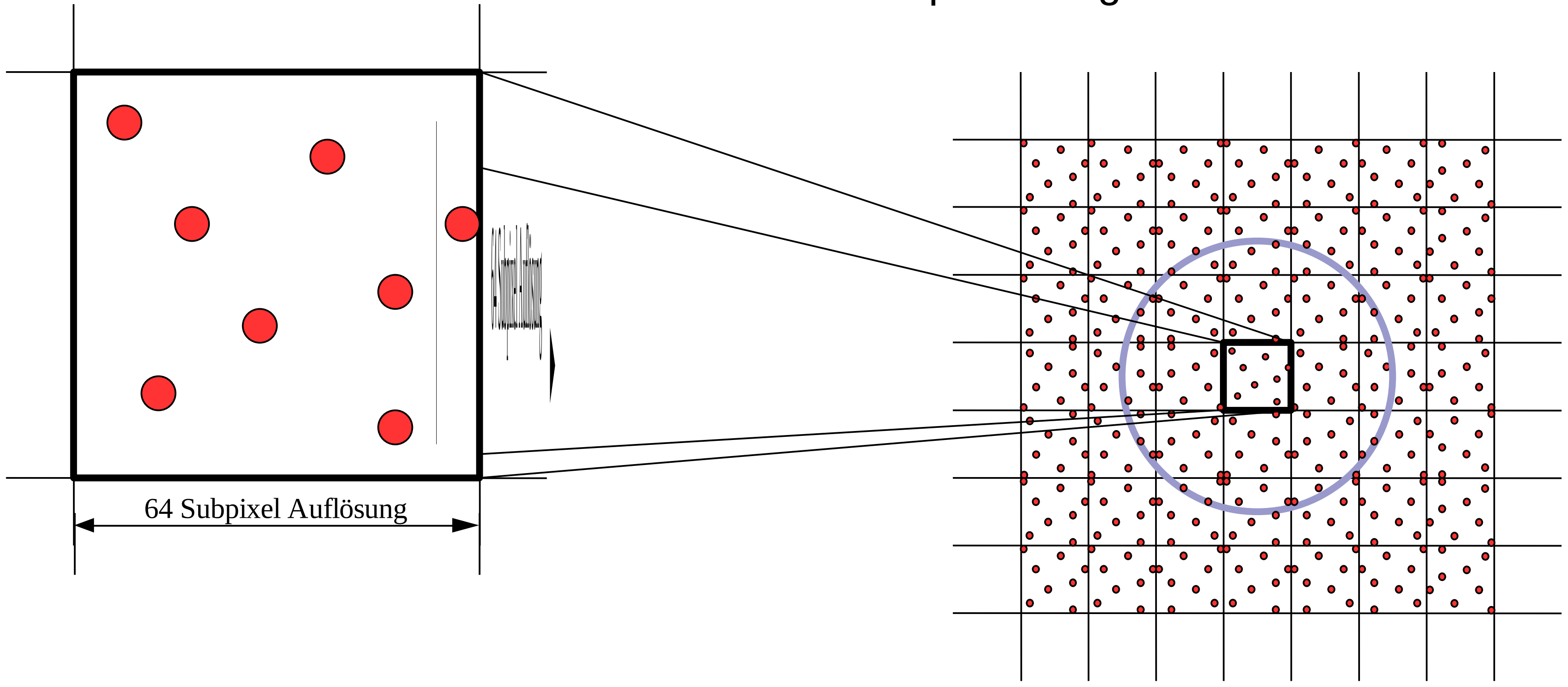

16 samples

Multisampling on XVR 4000

Courtesy of C. Gonzalez, Sun Microsystems

1) 1-16 Samples, random distribution that changes with each pixel, calculated in 64x64 subpixel area

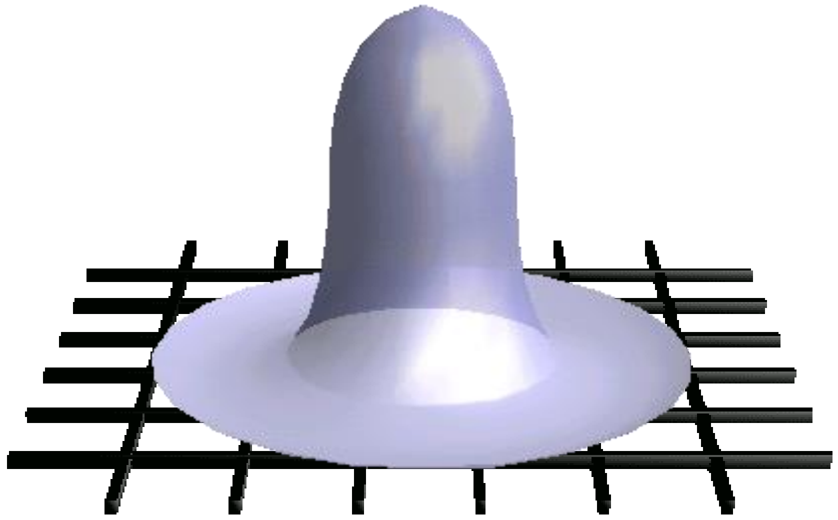
2) Samples are collected and filtered within an area shaped as circle and 5x5 pixels large



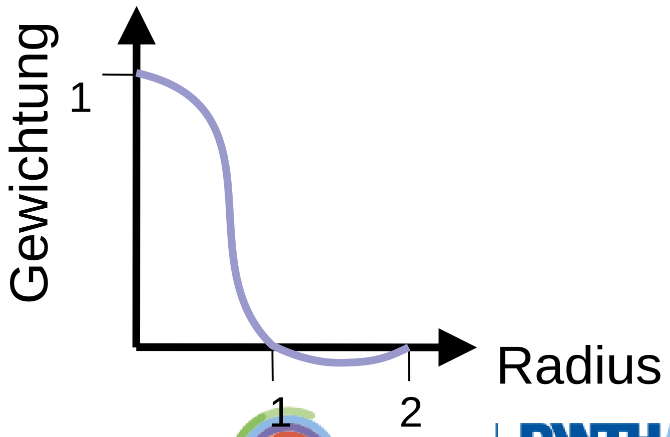
Multisampling on XVR 4000 (cont.)

Courtesy of C. Gonzalez, Sun Microsystems

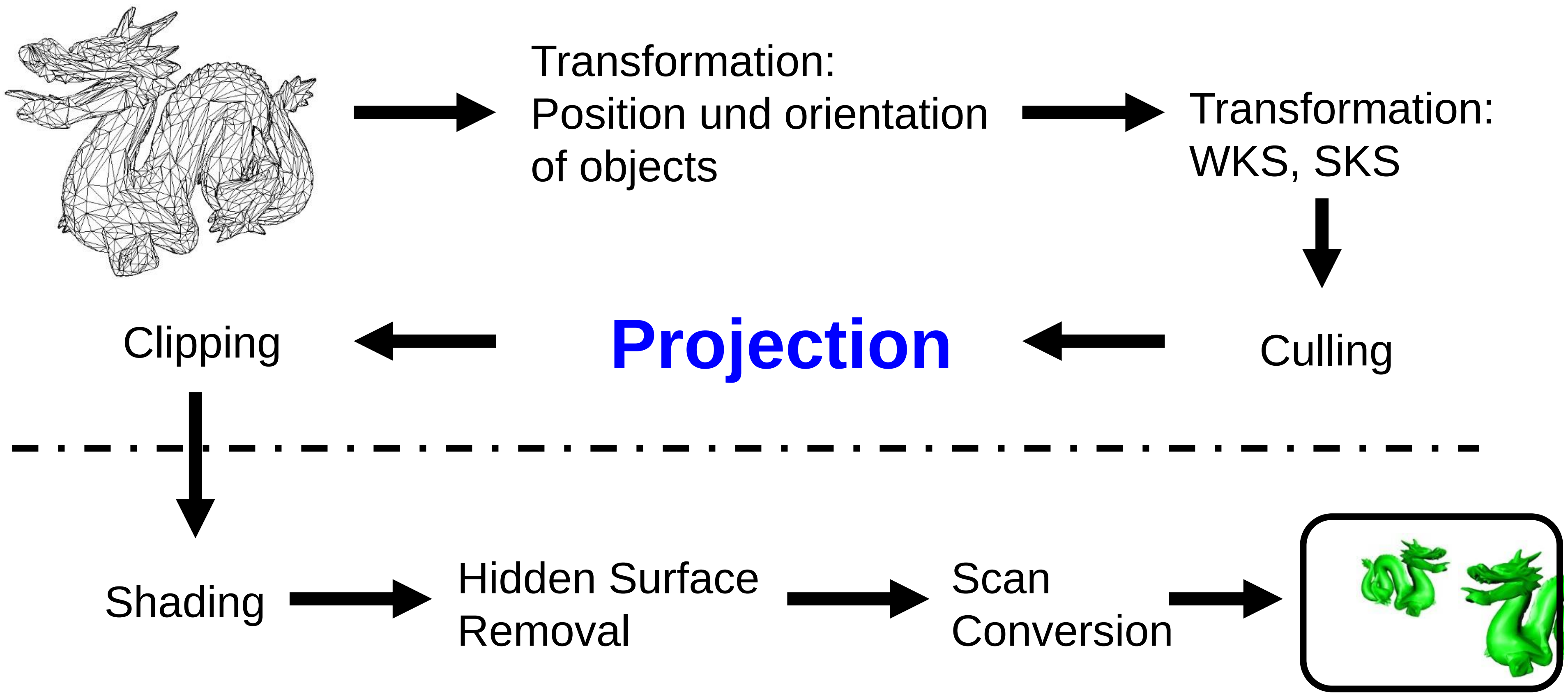
3) All samples within the circle are weighted, normalized and accumulated



4) The filter curve is programmable. User can adapt filter characteristics to the application



Overview – Rendering Pipeline



Topics – Basics in Computer Graphics / Rendering Pipeline

- **Rendering Pipeline**
- **Representation of rigid objects**
- **Transformations**
- **Culling**
- **Projection**
- **Clipping**
- **Hidden Surface Removal**
- **Shading**
- **Scan Conversion**
- **Texture Mapping**
- **Graphics Hardware**
- **Ray Tracing *(optional topic)***

Texture Mapping – Motivation

Texture Mapping: Usage of textures in CG (Catmull 1974)

Increase realism of virtual scenes:

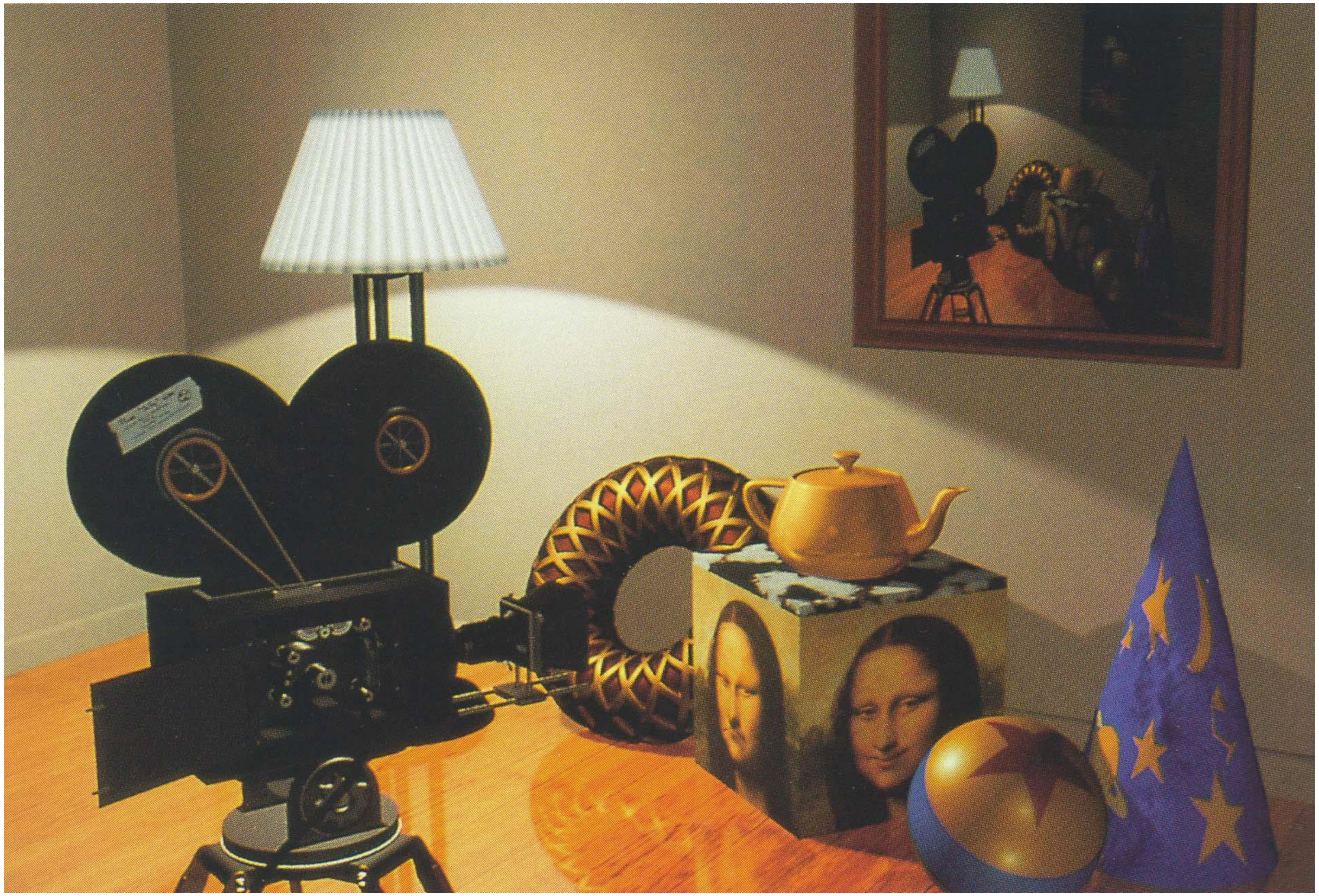
- Render the structure of surfaces without increasing geometry complexity
- Integrate real photos into the virtual scene
- Simulate mirror reflections without ray tracing
- Render semi-transparent surfaces (e.g., glass)

Scientific Visualization

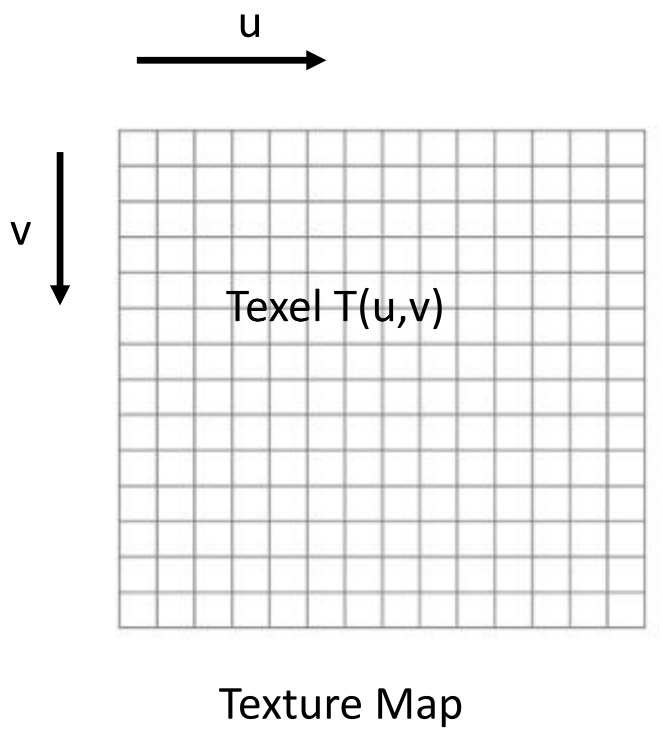
- LIC (linear integral convolution)
- Imaging in medicine: Render volume by textures

Texture Mapping – Modulation of Color

Picture: Foley et al.

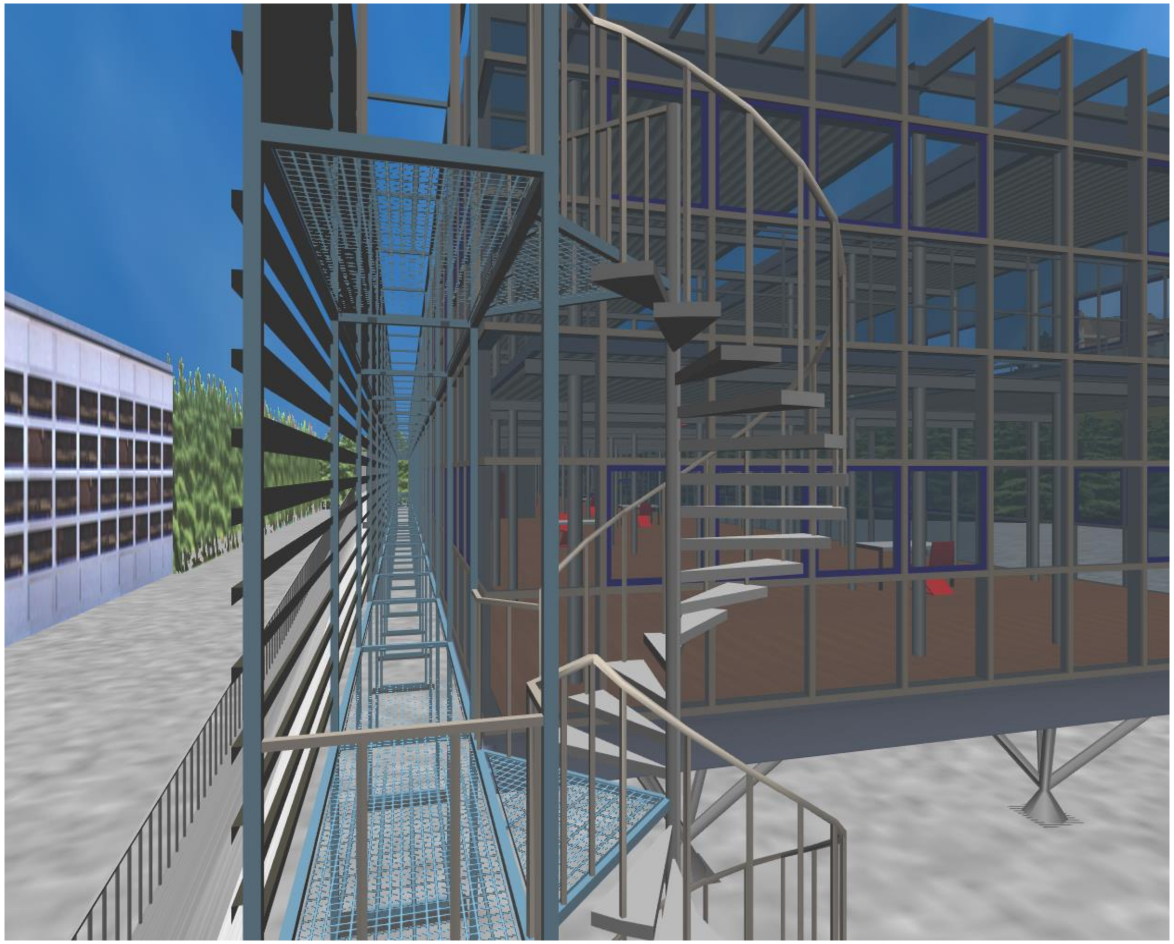


Texture Mapping – Parameter Modulation



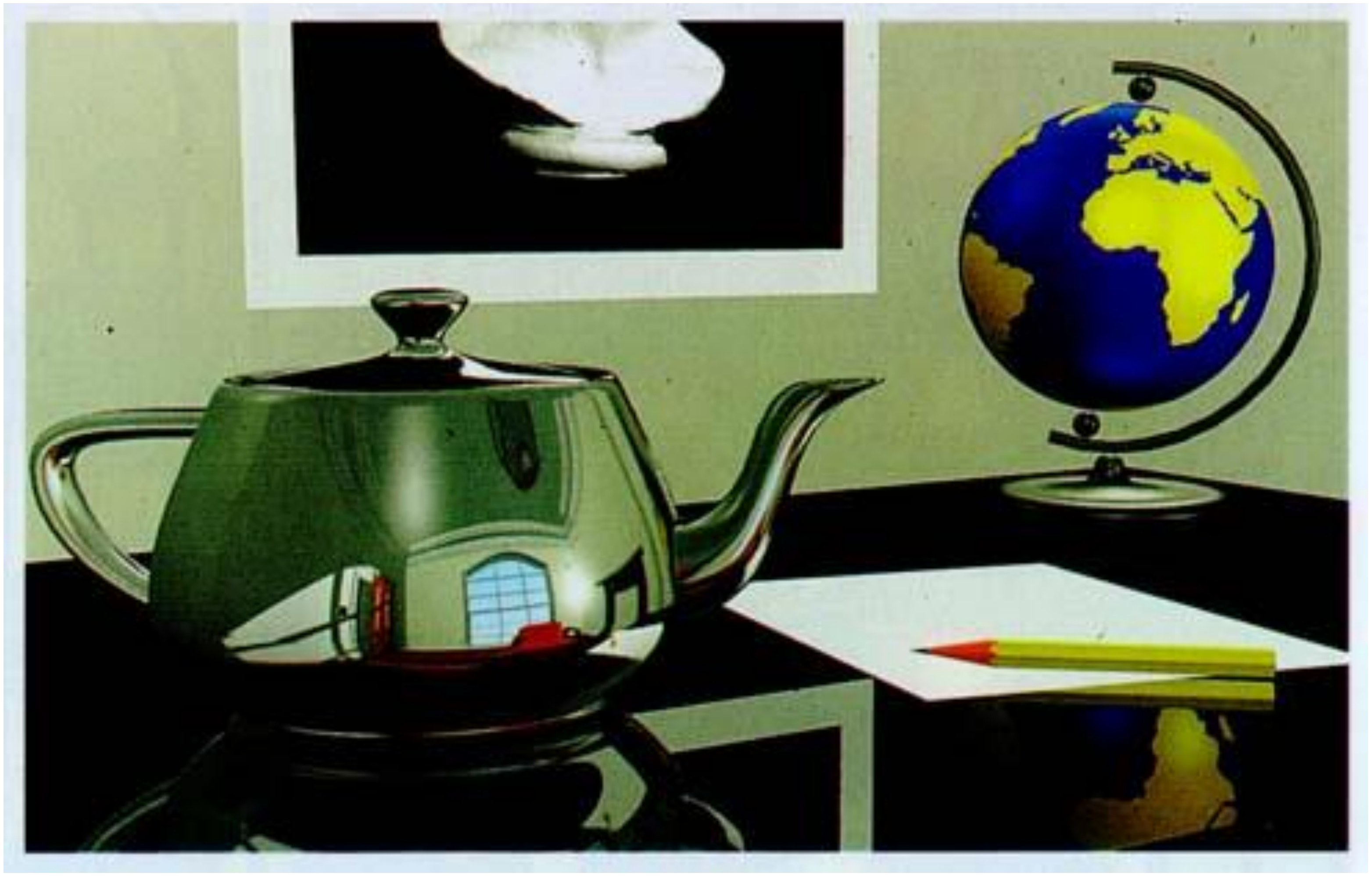
- Color
  - Functional or procedural
  - Randomized
  - Photo
- Specular and diffuse component (Environment Mapping)
- Transparency (α- Blending)
- Perturbation of the normal vector (Bump Mapping)

Texture Mapping – Alpha Blending

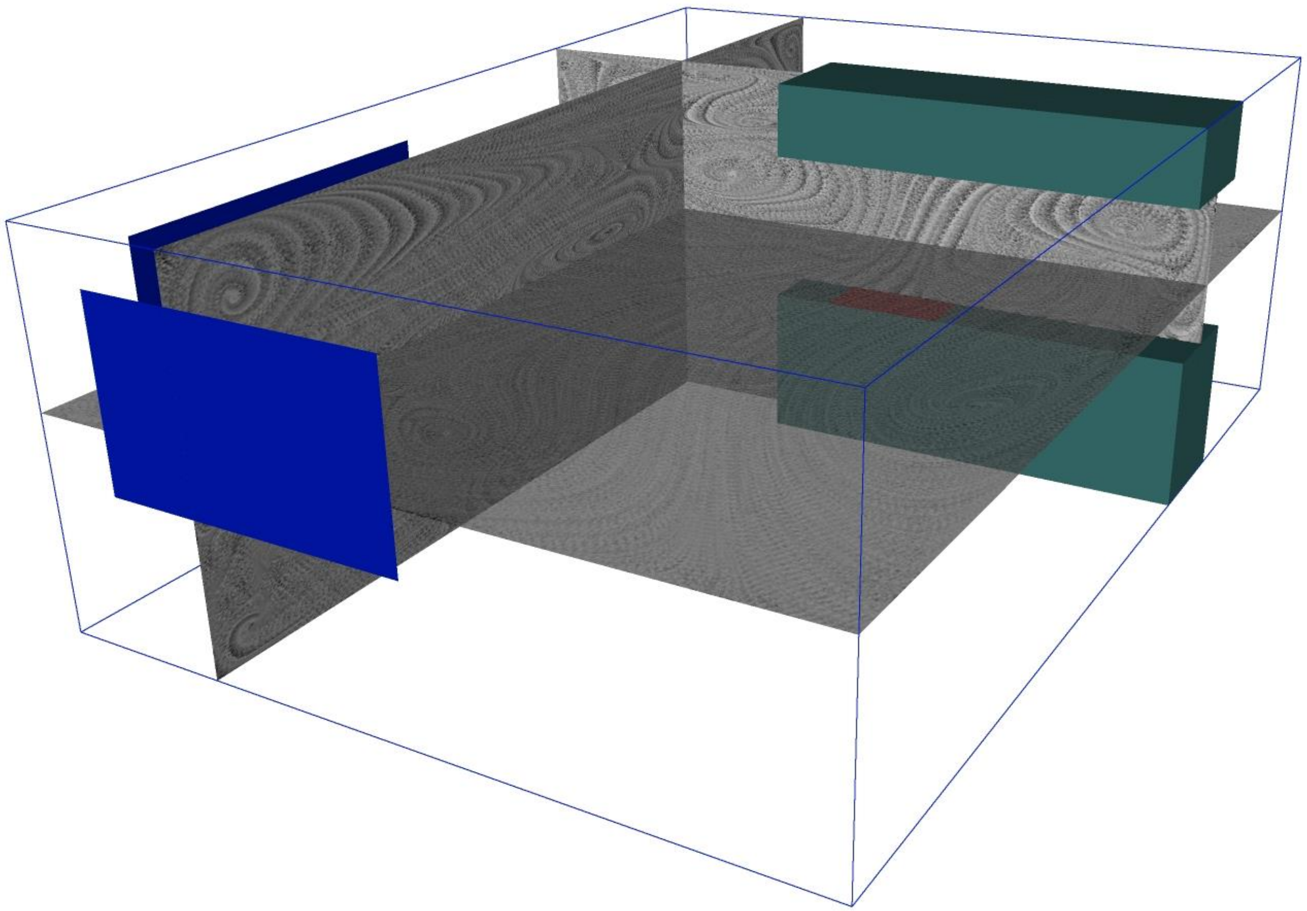

Virtual building at Schinkel Straße, RWTH Aachen University
(Generated by IT Center in 1998)

Texture Mapping – Environment Mapping

Picture: Watt



Texture Mapping – Linear Integral Convolution

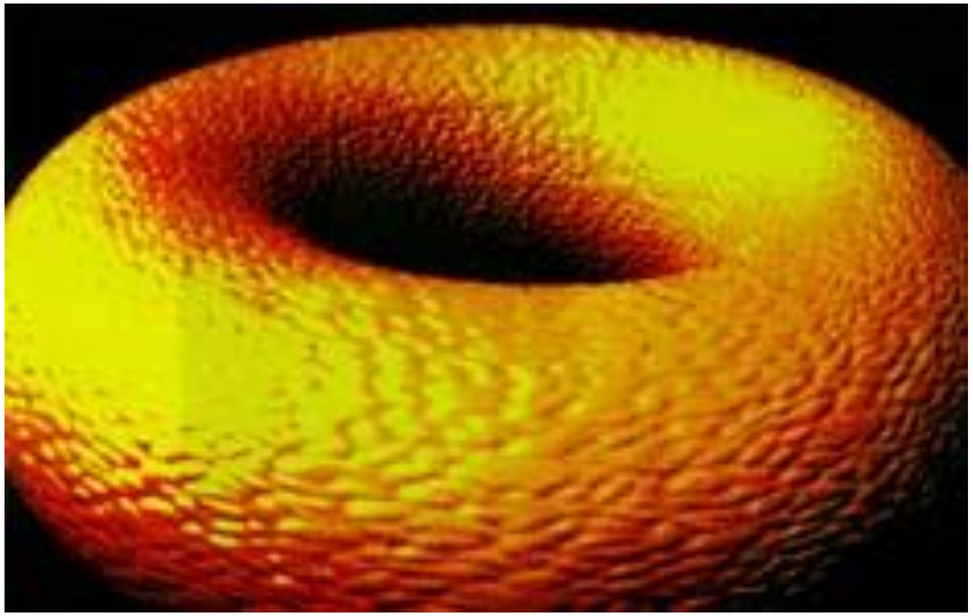


Simulation of air flow within a kitchen
(here: faster method without convolution – "integrate-and-draw")

Picture: IT Center RWTH

Texture Mapping – Bump Mapping

Pictures: Watt



Texture Mapping – S-Mapping

T $(u,v) \rightarrow$ T´(x_i, y_i, z_i): „S-Mapping“

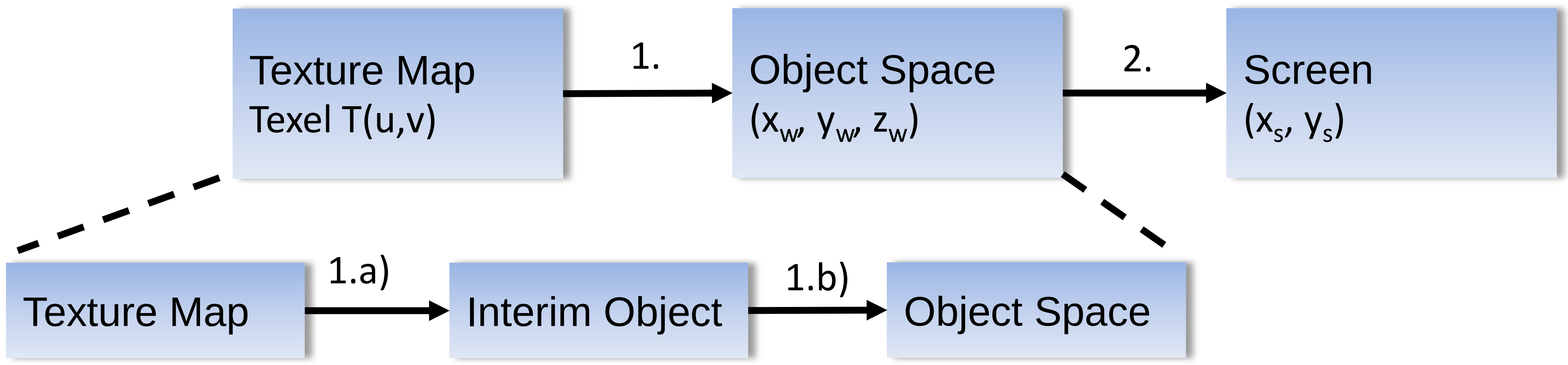
Interim objects

- enclose the virtual object
- simple mapping functions

- Arbitrarily oriented plane
- Cylinder (without top and bottom sides)
- Sphere
- Cube (No distortion, but clipping necessary)

From Texture Map to Pixels

1. Surface parameterization
2. Projection (interpolation, see Shading)

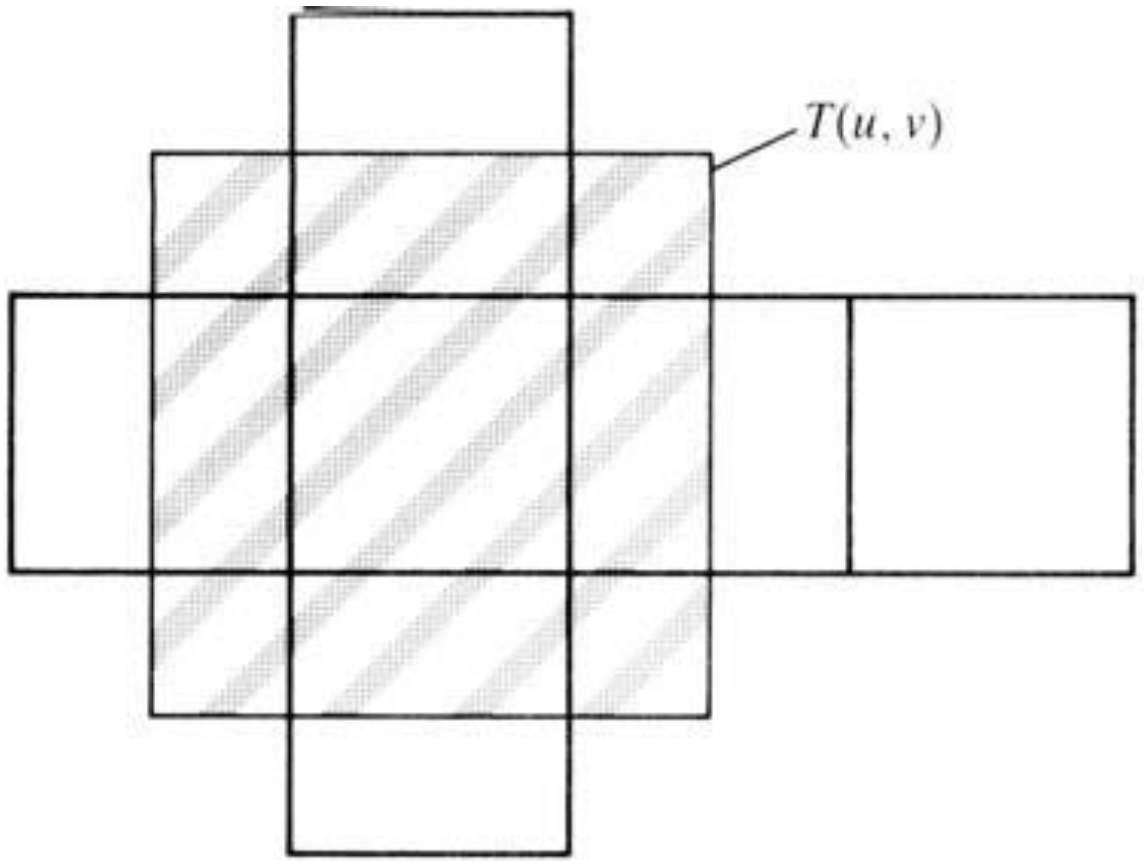


Use 2-phase parameterization to avoid distortions for polygon models:

a) S-Mapping

b) O-Mapping

S-Mapping on a Cube



S-Mapping Example: Cylinder

$$S_{Cylinder} : (\Theta, h) \rightarrow (u, v) = \left(\frac{r}{c}(\Theta - \Theta_0), \frac{1}{d}(h - h_0) \right), \text{ with}$$

Θ_0, h_0 : Texture position

c, d : scaling factors

r : Cylinder radius

$\Theta : [0, 2\Pi]$

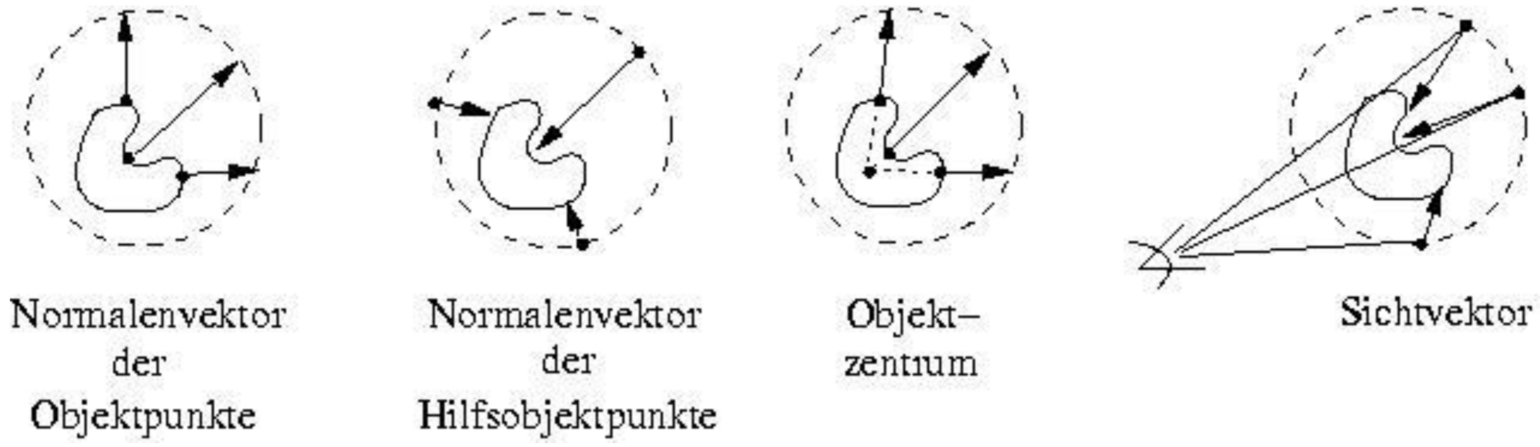
$h : [0, \text{cylinder height}]$

Texture Mapping – O-Mapping

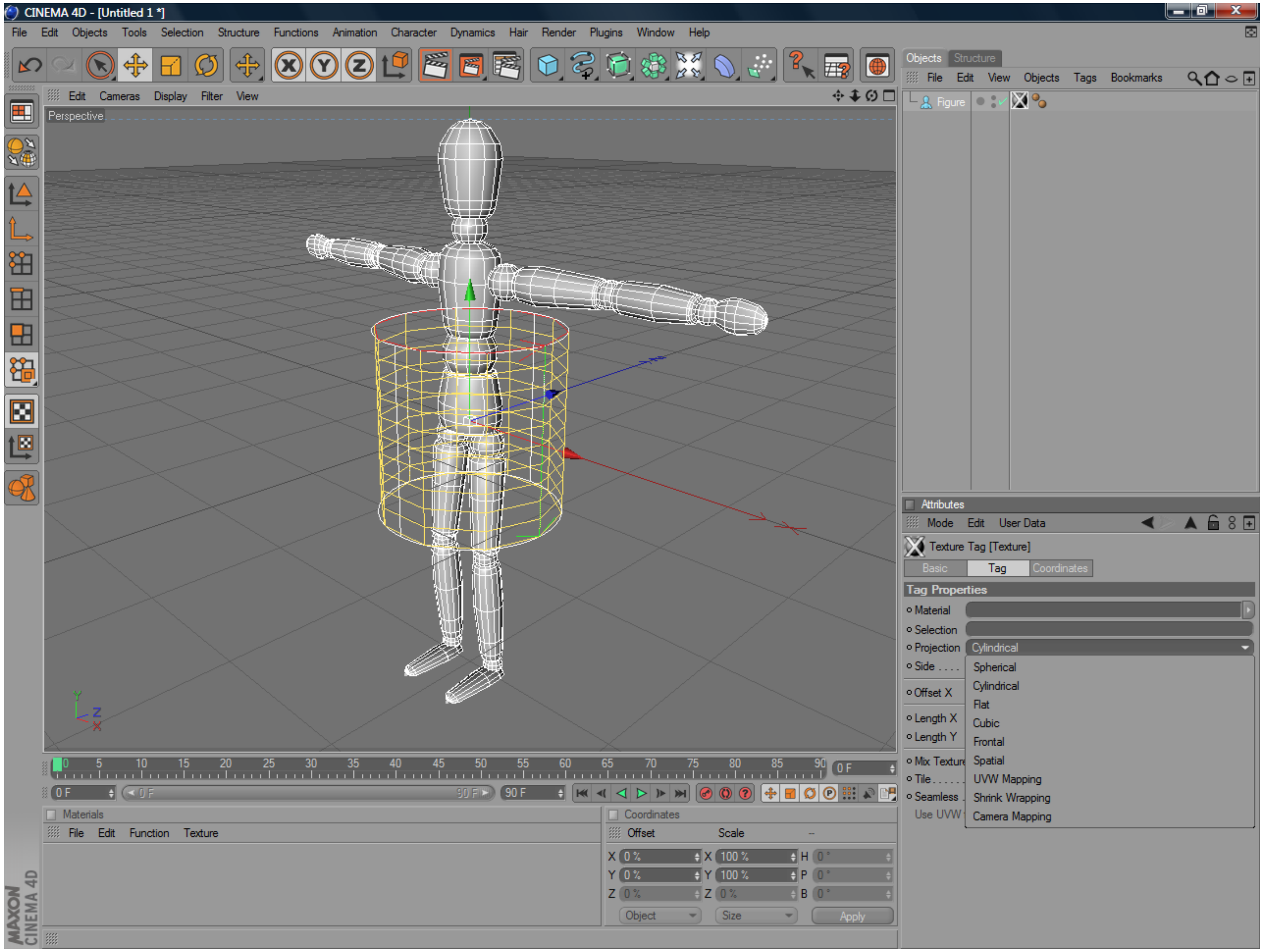
$$T'(x_i, y_i, z_i) \rightarrow O(x_w, y_w, z_w)$$

4 strategies:

1. Intersection point of the surface normal vector at every position (x_w, y_w, z_w) and T$'(x_i, y_i, z_i)$
2. Intersection point of the interim object at every position with the object surface
3. Projection of the object midpoint onto the interim surface
4. Mirror reflection of a sight vector at the interim object. Texture seems to walk over the object surface when the viewer moves (Environment Mapping).

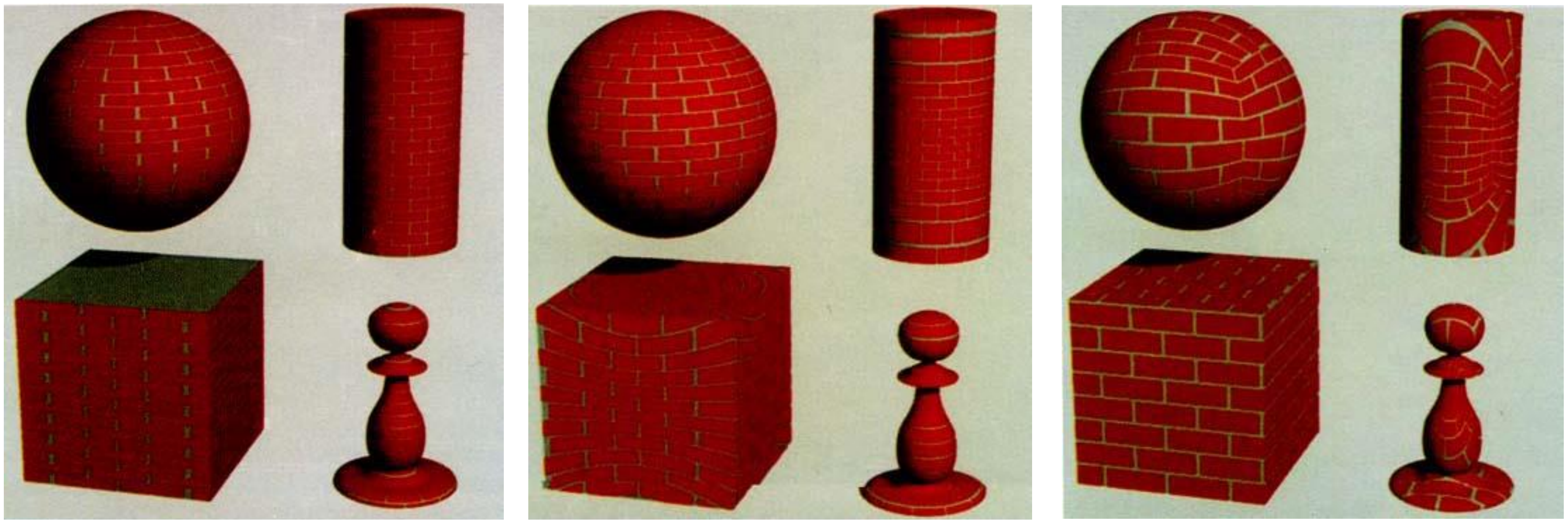

Normalenvektor der Objektpunkte
Normalenvektor der Hilfsobjektpunkte
Objekt–zentrum
Sichtvektor

Texture Mapping – Cinema 4D

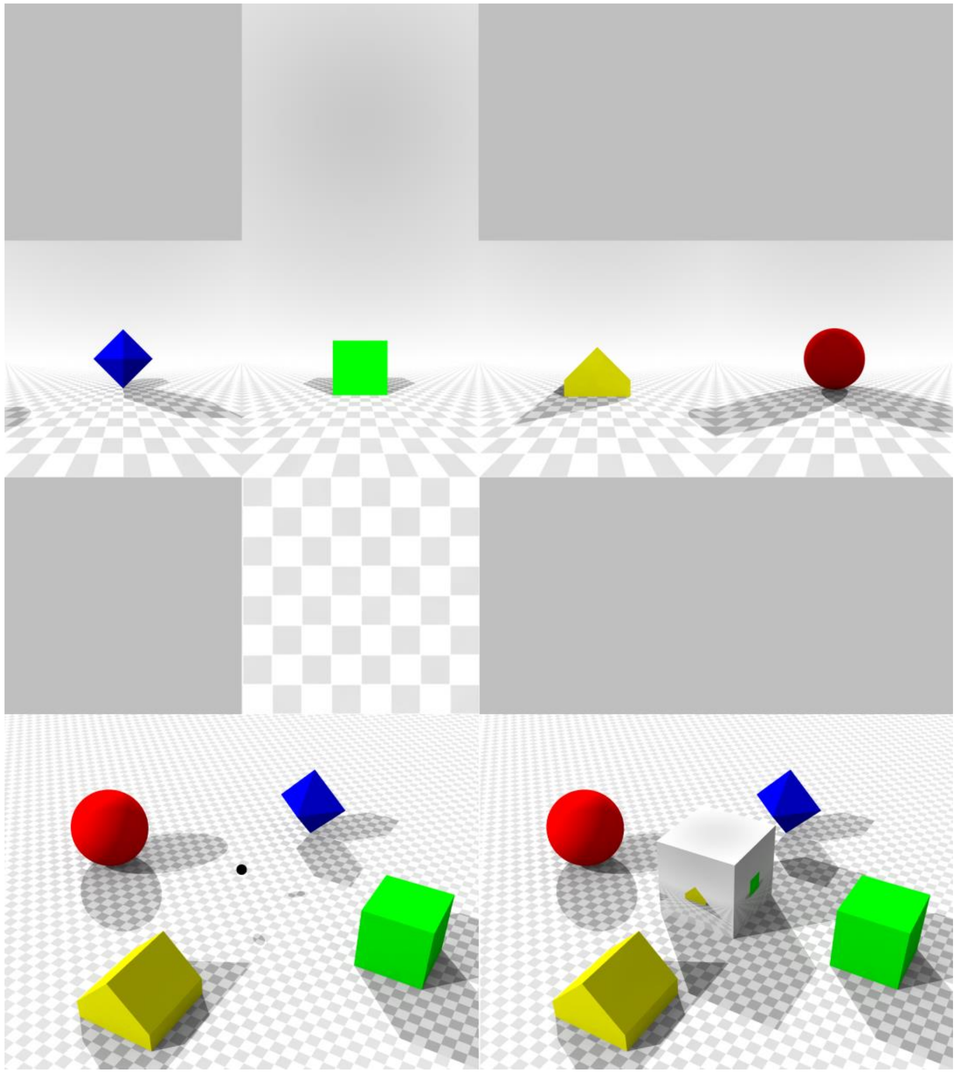


S-Mapping – Effects of Different Interim objects

Pictures: Watt

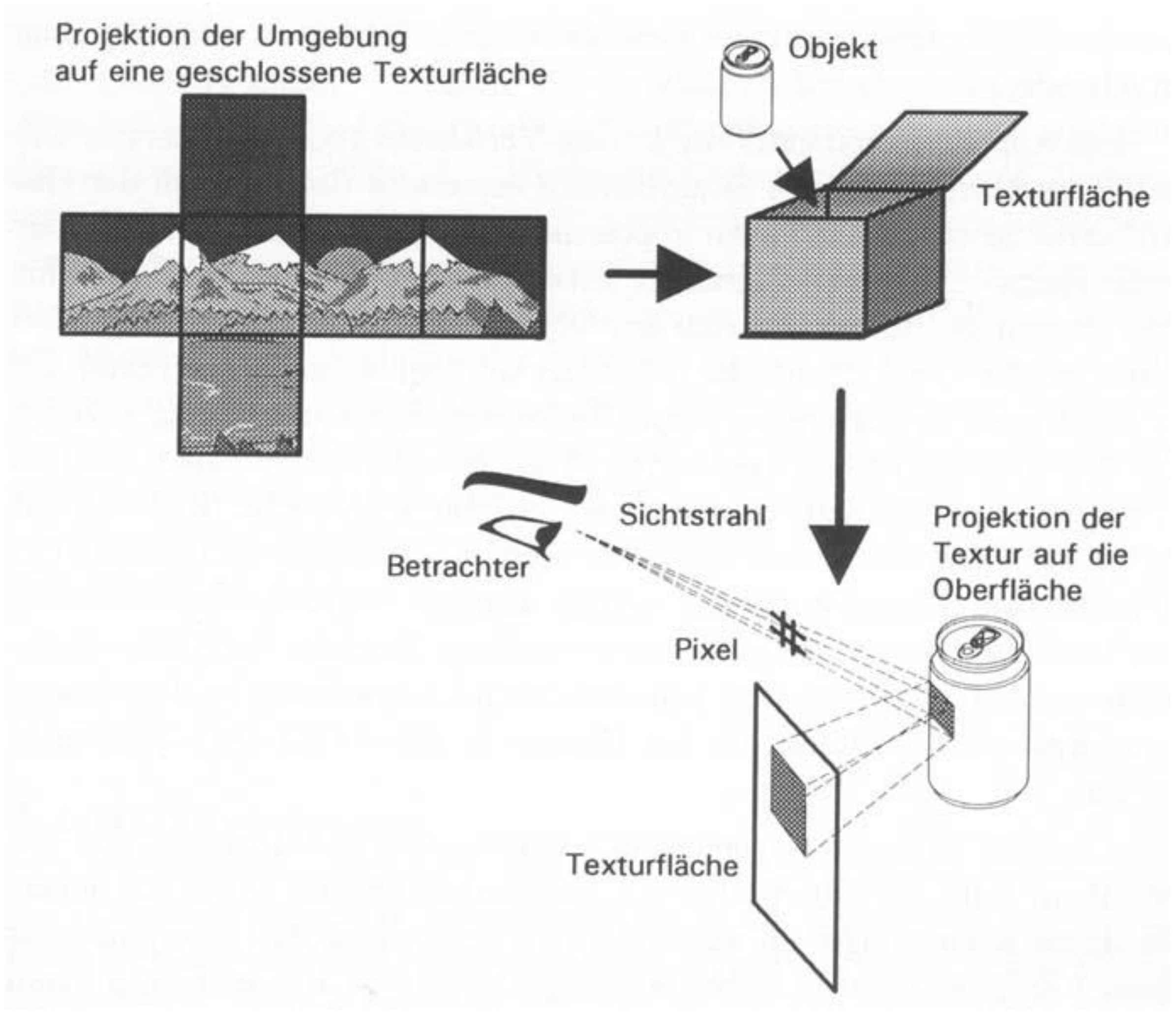


Cube Mapping



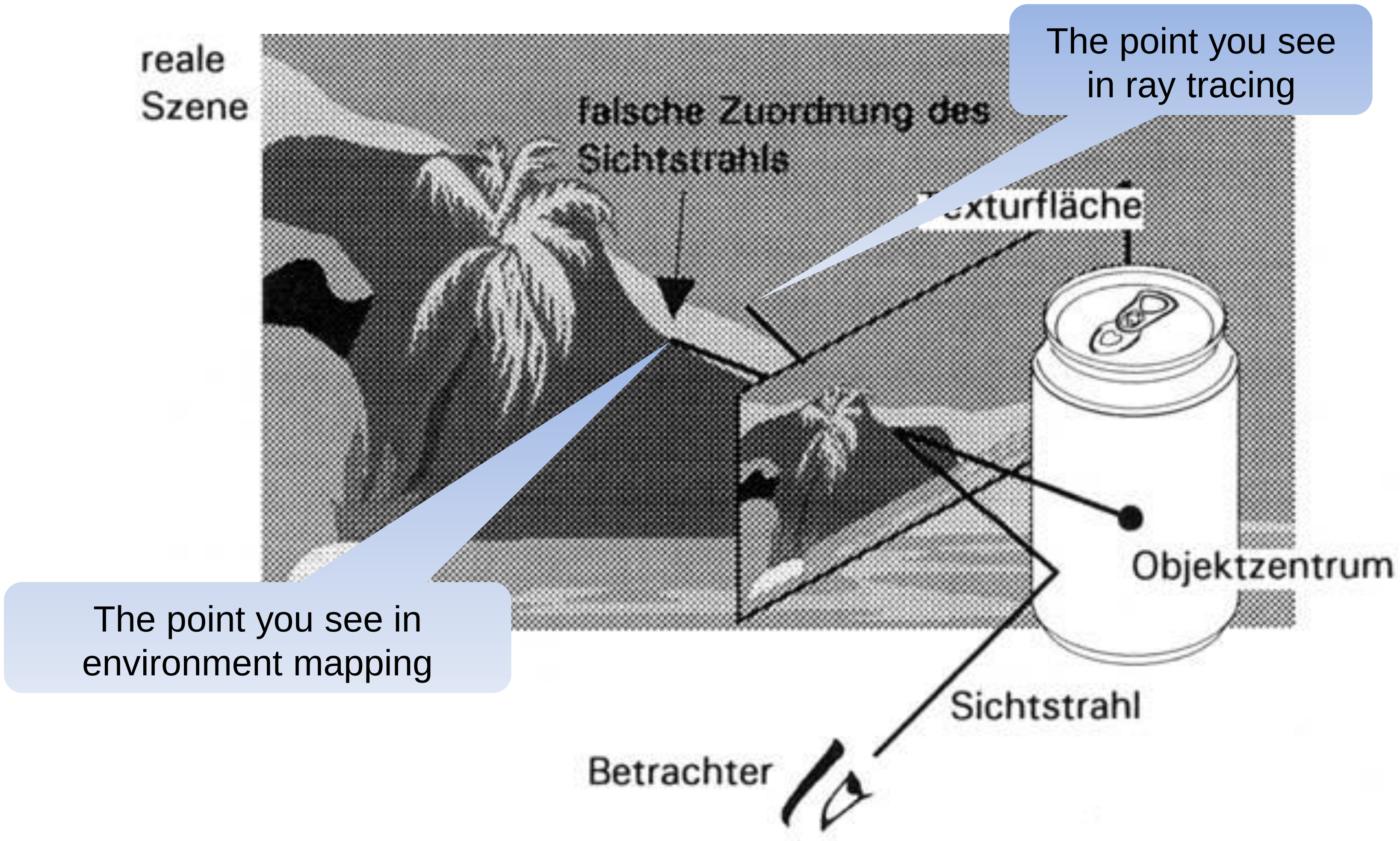
The Principle of Environment Mapping

Picture: Tönnies



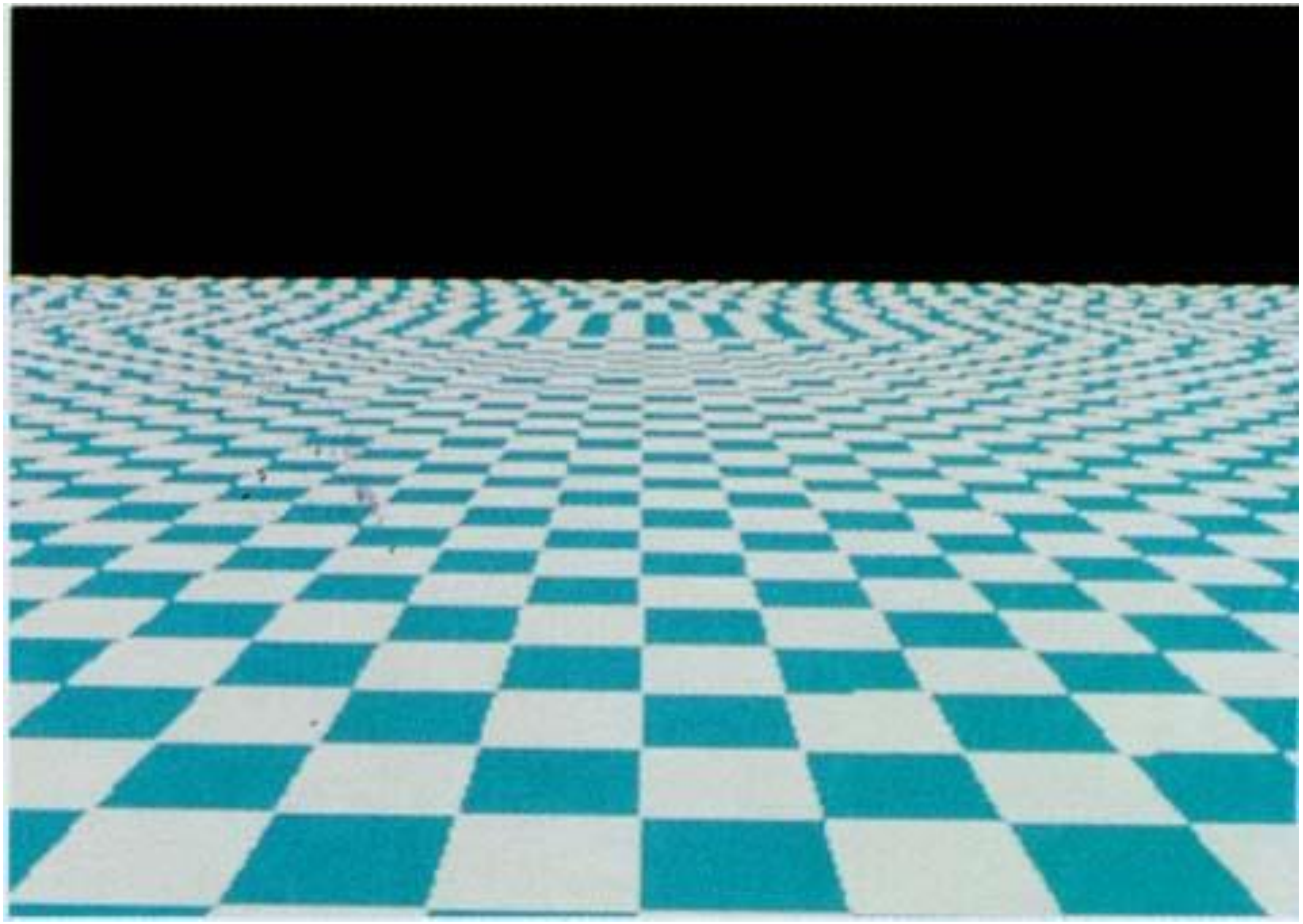
Environment Mapping vs. Ray Tracing

Picture: Tönnies



reale Szene
falsche Zuordnung des Sichtstrahls
Texturfläche
Objektzentrum
Sichtstrahl
Betrachter
The point you see in ray tracing
The point you see in environment mapping

Texture Mapping – Aliasing Example



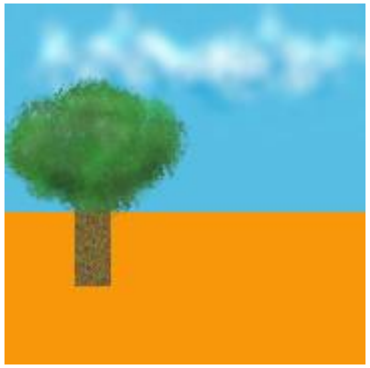
Texture Mapping – Aliasing Effects

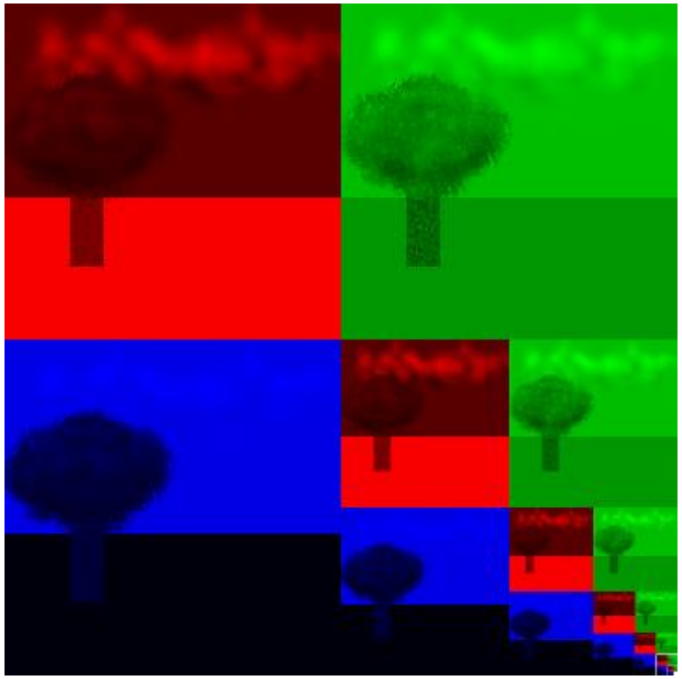
2 extreme situations:

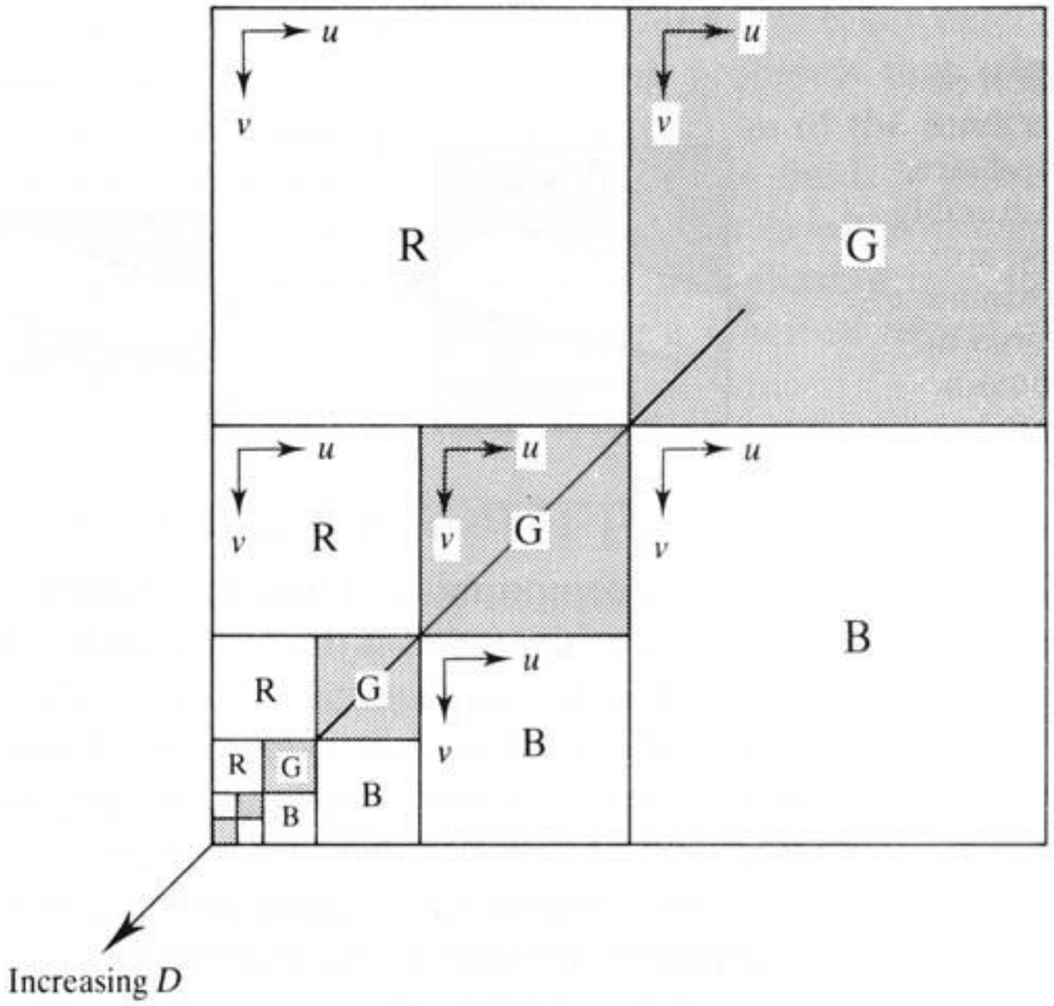
1. Viewer is very close to the textured object: Only one texel for many pixels (tiling effect)
2. Textured object is very far from the viewer: Many texels for only one pixel (see example on last slide)

ad 1/2.	Texture hierarchies and Mip Mapping
ad 2.	Shrink Wrap Method and Inverse Mapping

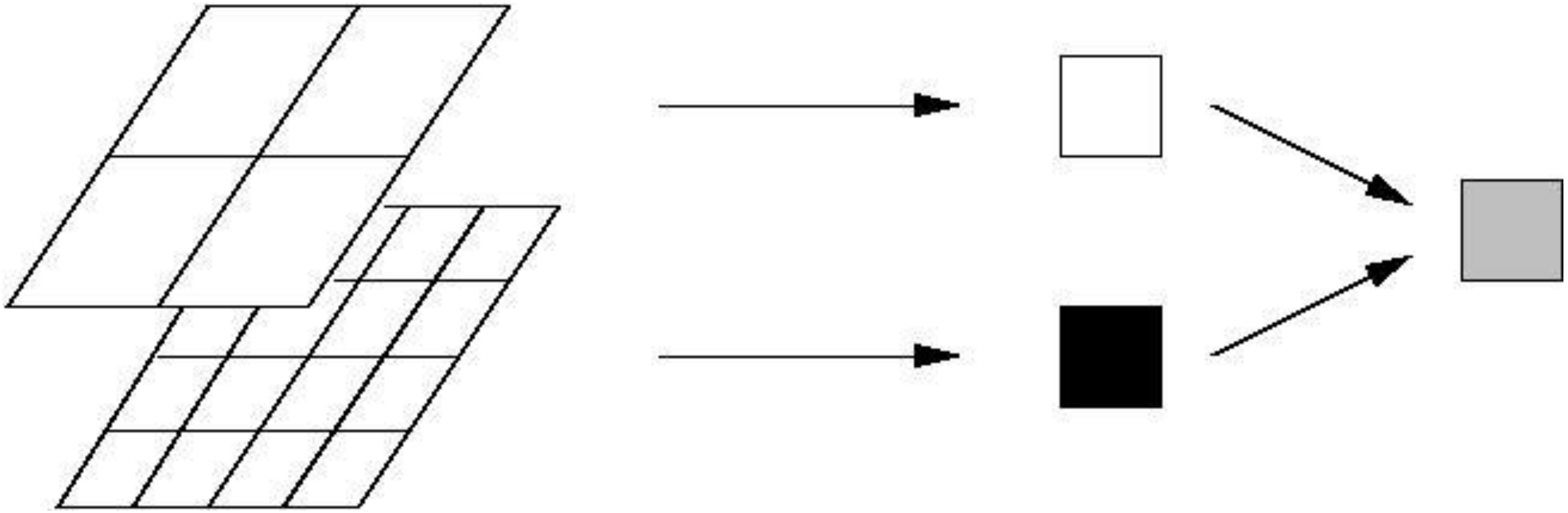
Texture Hierarchies





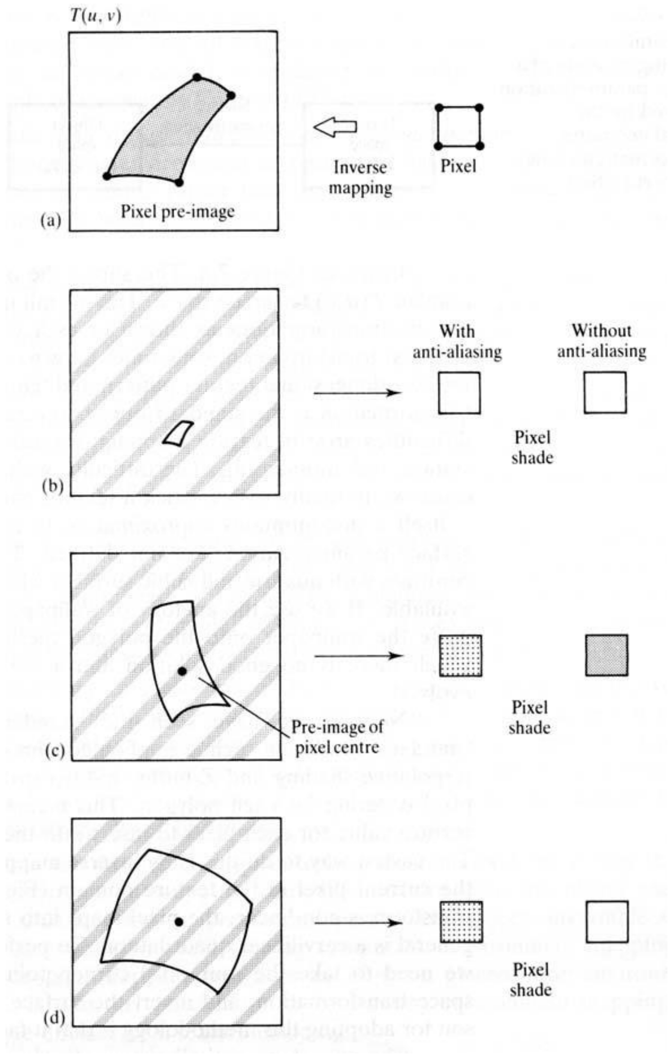


Mip Mapping – Trilinear Interpolation



Shrink Wrap Method & Inverse Mapping

- Start from the 4 pixel corners (Inverse Mapping)
- Transform all 4 corners into texture coordinates
- Integrate over the surface in the texture map that is covered by the pixel



Topics – Basics in Computer Graphics / Rendering Pipeline

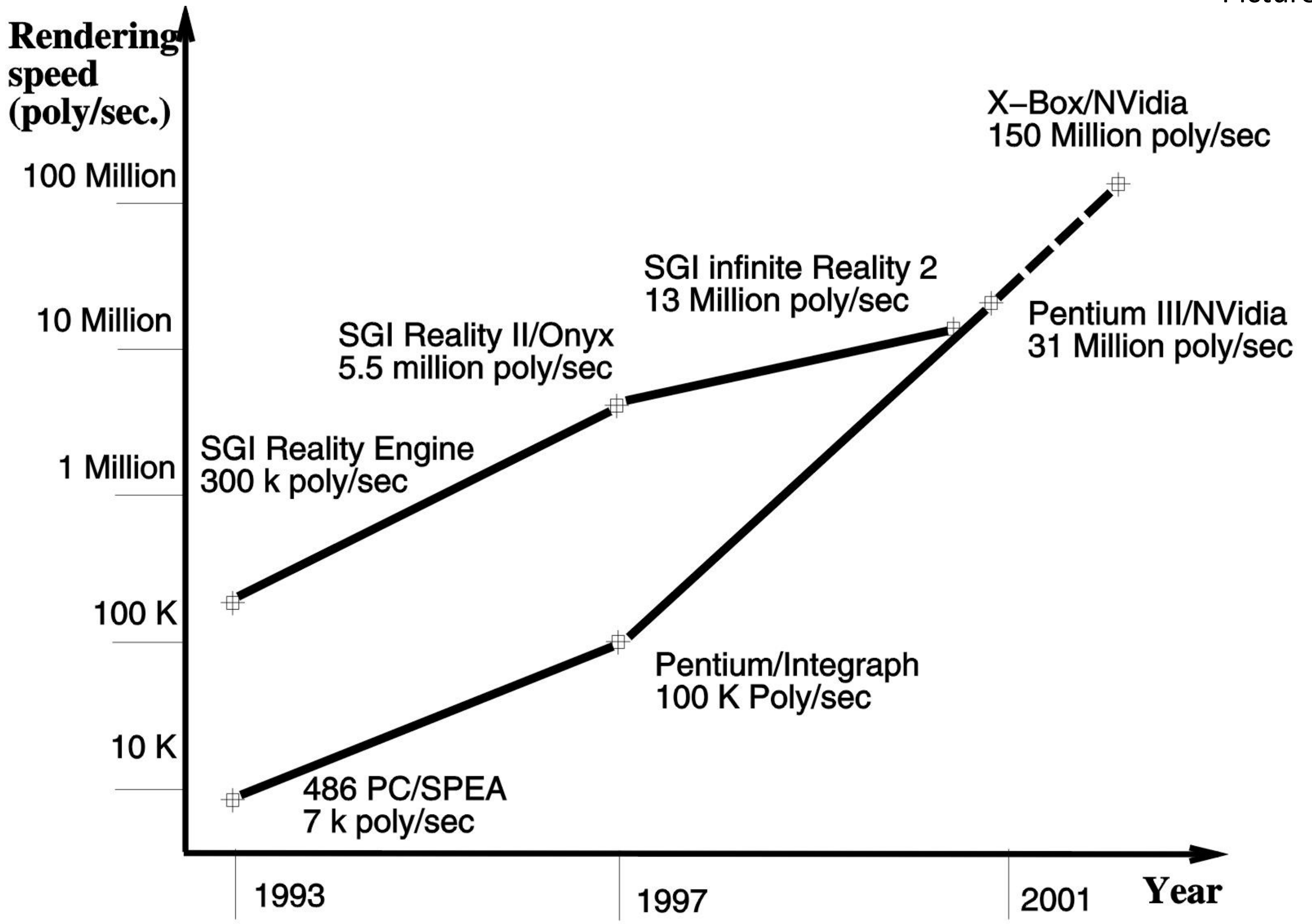
- Rendering Pipeline
- Representation of rigid objects
- Transformations
- Culling
- Projection
- Clipping
- Hidden Surface Removal
- Shading
- Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing *(optional topic)*

How to Measure Graphics Performance

- **Speed:**
  - Number of polygons per second (Attention: Size and shape of polygons, shading method, ...)
  - Fill Rate
  - Data transport: Graphics hardware – CPU – main memory
  - Benchmarks: ViewPerf, ... (see WWW)
- **Quality:**
  - Memory – Frame Buffer (size, color depth), Z-Buffer (depth), Texture Memory, ...
  - Texture Mapping Features
  - Hardware Anti-Aliasing
  - Special Features: Stereo, Vertex & Pixel Shaders, ...
- **Flexibility:**
  - Configuration Interface
  - Number of resolutions and screens supported, ...

History of Graphics Performance

Picture:Burdea et al.



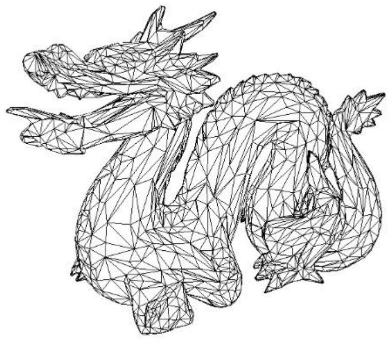
How to make graphics chips faster…

- Chip development, processor clock speed
- Fast and larger busses (PC:AGP, PCI Express)
- Add more memory: Resolution, color depth, z-buffer depth, amount of textures, anti-aliasing
- Pipelining
  - „macroscopic“ (Rendering Pipe)
  - onChip (float-multiplication)
- „Real“ parallel (SIMD,...)
  - „macroscopic“ (more Chips)
  - onChip (more ALUs, …)

2 strategies:

- Geometry on CPU, scan conversion on graphics hardware
- Both on graphics hardware

Overview – Rendering Pipeline



Transformation:
Position und orientation
of objects

Transformation:
WKS, SKS

Culling

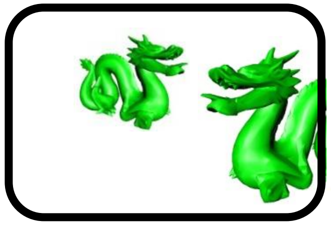
Projection

Clipping

Shading

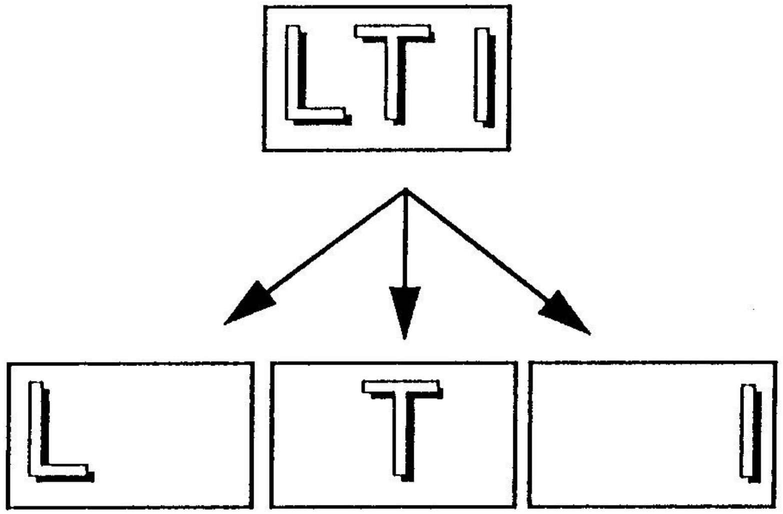
Hidden Surface
Removal

Scan
Conversion

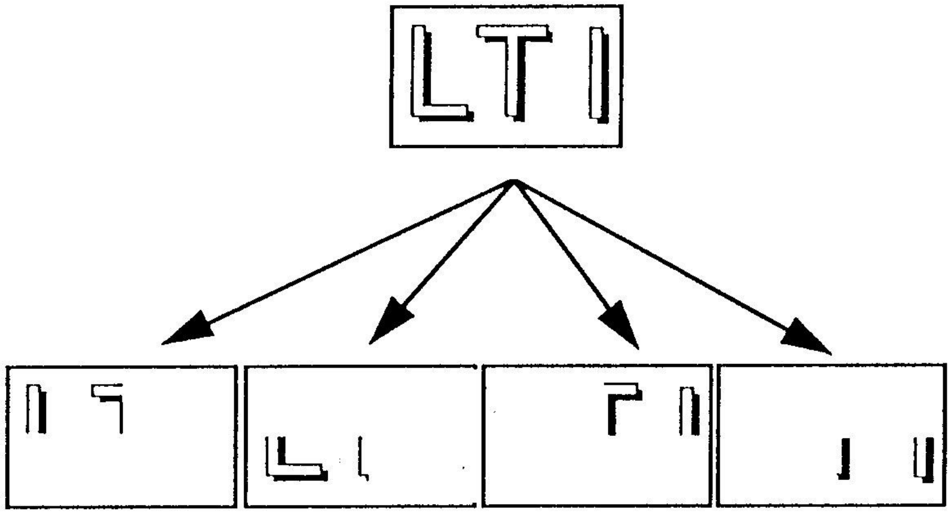


Composition vs. Partition

Picture: Kraiss, Technische Informatik, RWTH Aachen



Composition



Partition

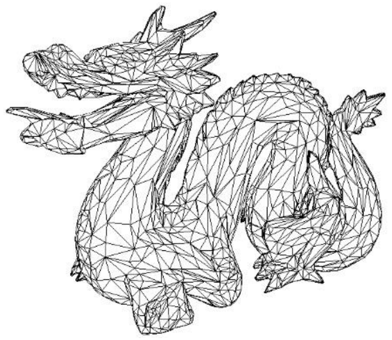
Topics – Basics in Computer Graphics / Rendering Pipeline

- Rendering Pipeline
- Representation of rigid objects
- Transformations
- Culling
- Projection
- Clipping
- Hidden Surface Removal
- Shading
- Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing *(optional topic)*

Ray Tracing – Topics

- What is Ray Tracing?
- The Basic Algorithm
- Intersections
- Light and Reflection
- Reflection- and Transmission Rays
- Optimization
  - Quality
  - Speed

Overview – Rendering Pipeline



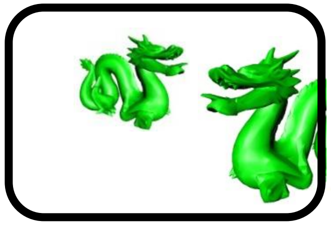
→ Transformation: Position und orientation of objects → Transformation: WKS, SKS

↓

Culling → **Projection** → Clipping

↓

Shading → Hidden Surface Removal → Scan Conversion →



Ray Tracing -The Idea

Whitted 1980:

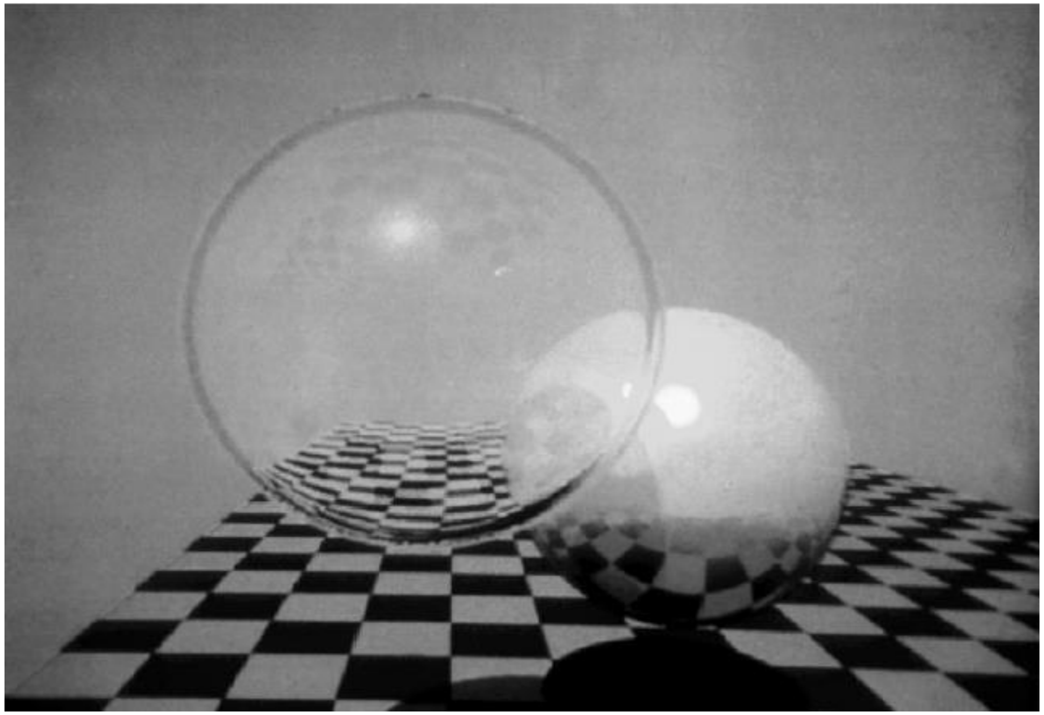
Create photorealistic images by following single light beams

Simulate the process of light distribution by the laws of ideal mirror reflection and refraction

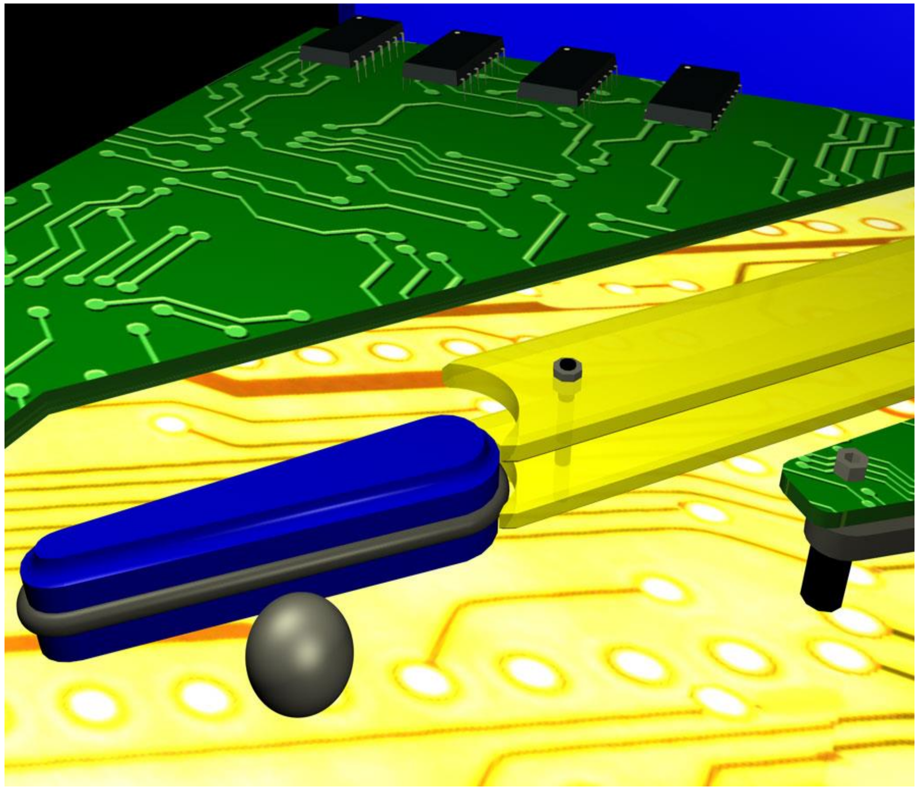
Root:

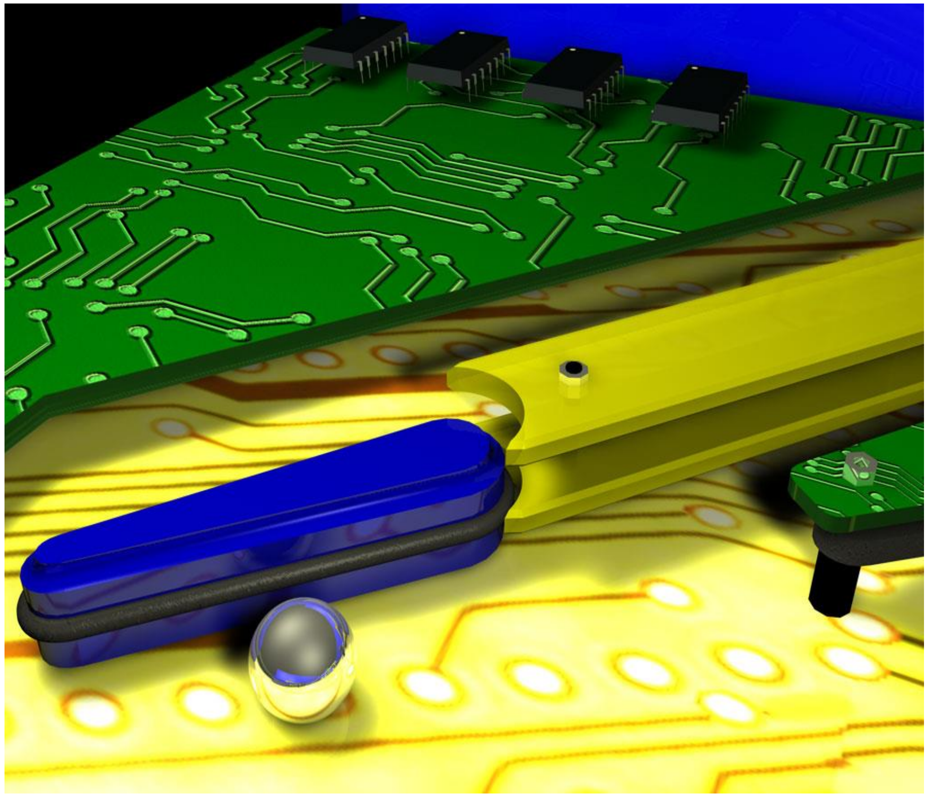
Ray optics in physics
(e.g., Descartes)

- Occlusion (HSR)
- Shadows
- Reflection
- Refraction



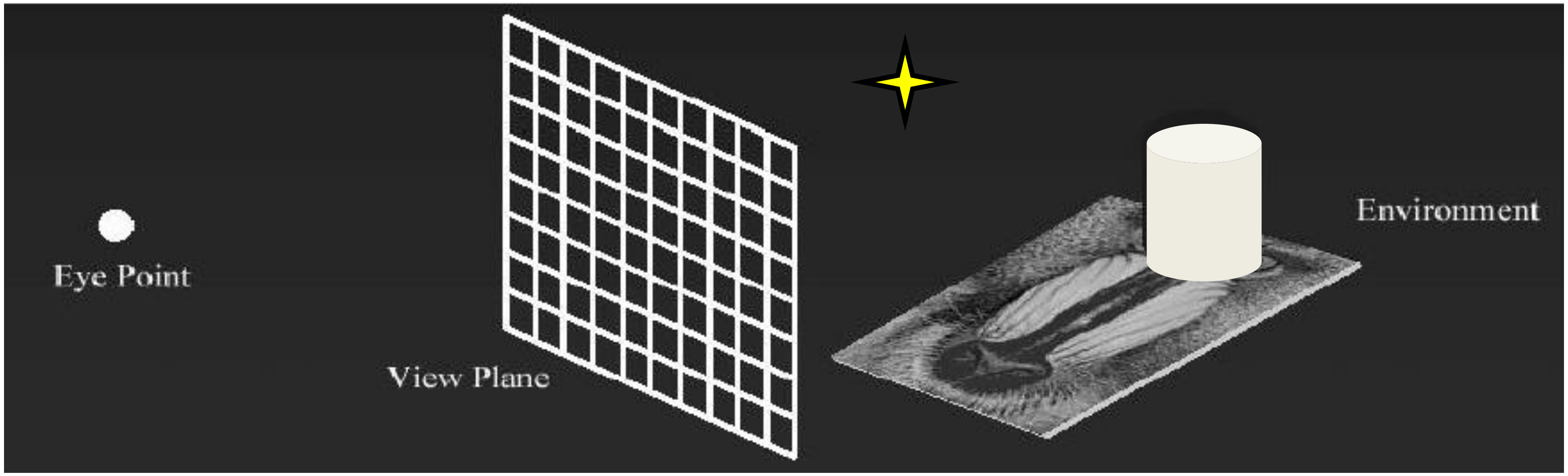
Gouraud Shading vs. Ray Tracing



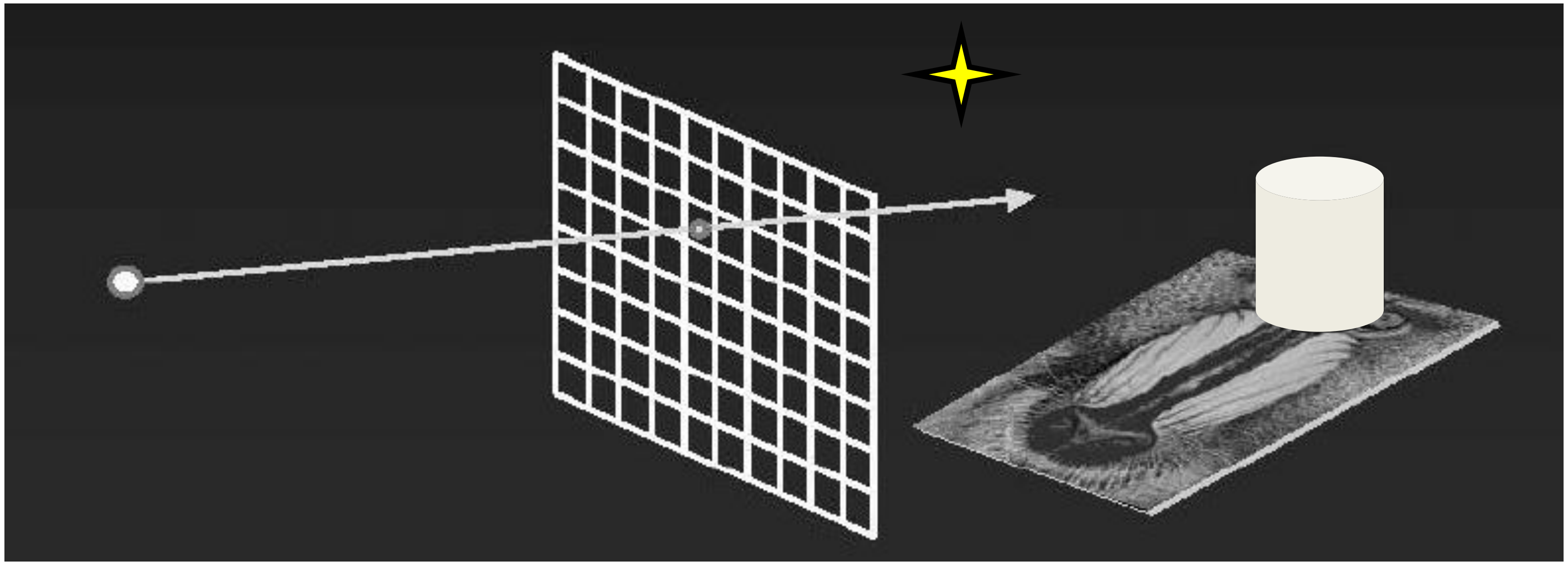


The starting situation

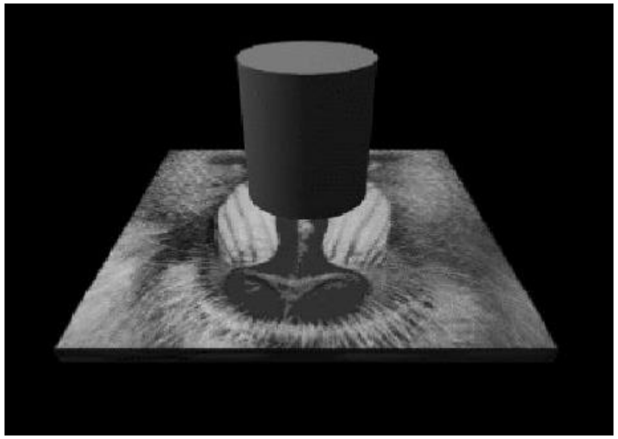
- 3-D scene
  - Objects
  - (Point) light sources
- Eye position
- Screen (viewing plane), Pixel



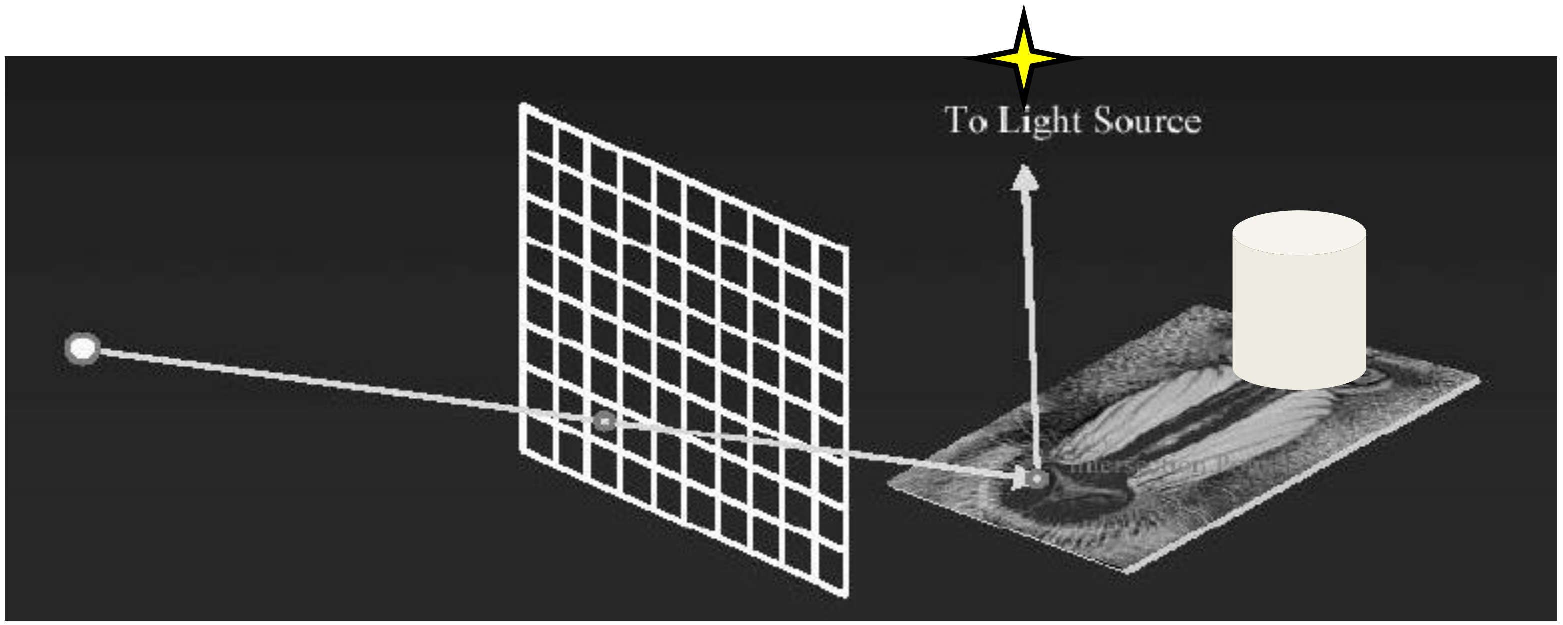
First Step: Ray Casting



- View rays through all pixels of the viewing plane
- Intersection point of the view ray with an object
- Appel (1968)
  Hidden Surface Removal

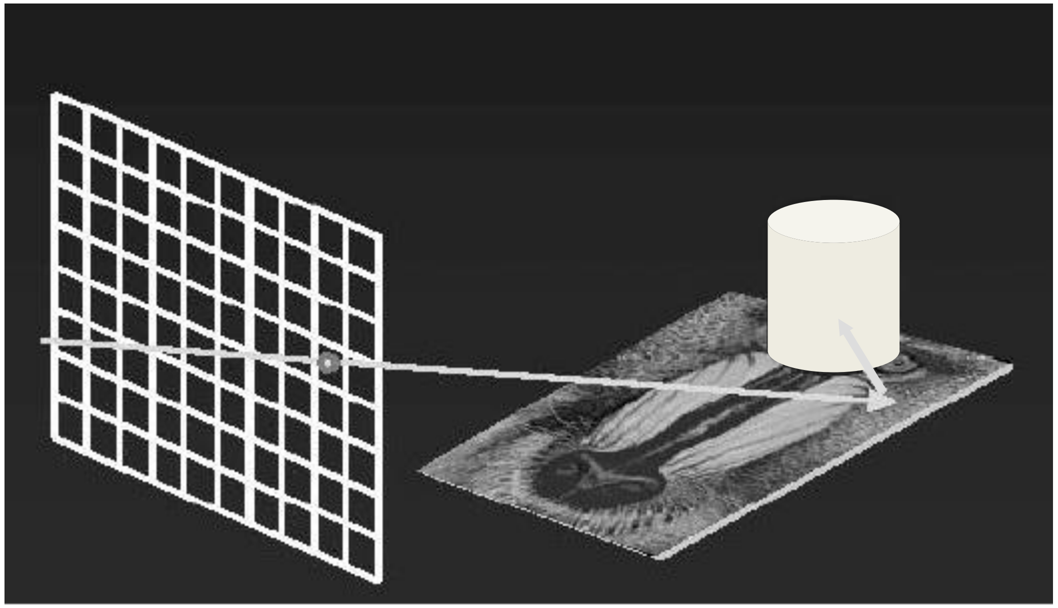


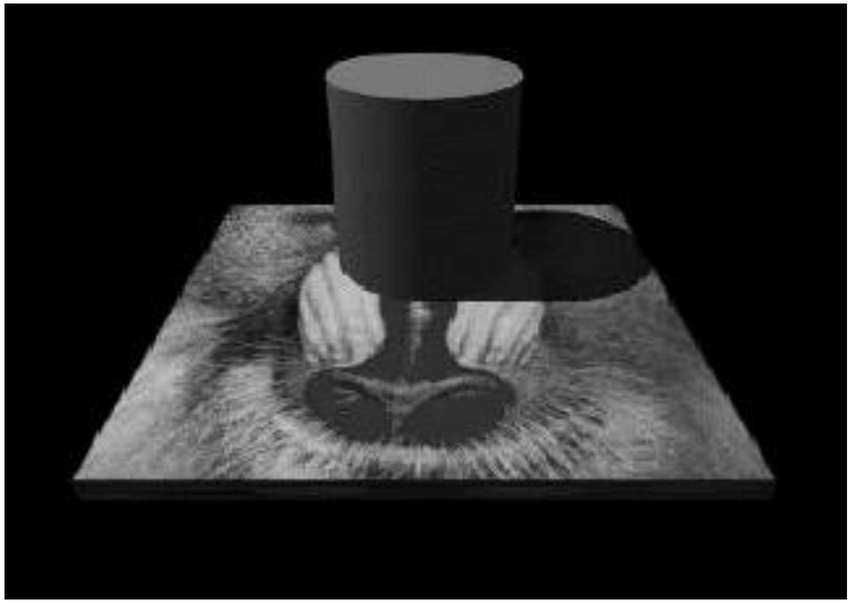
Shadow Feelers I



- „Shadow Feelers“: Rays from the intersection point to the light sources
- Calculate intensity at the intersection point (e.g., Phong Reflection Model)

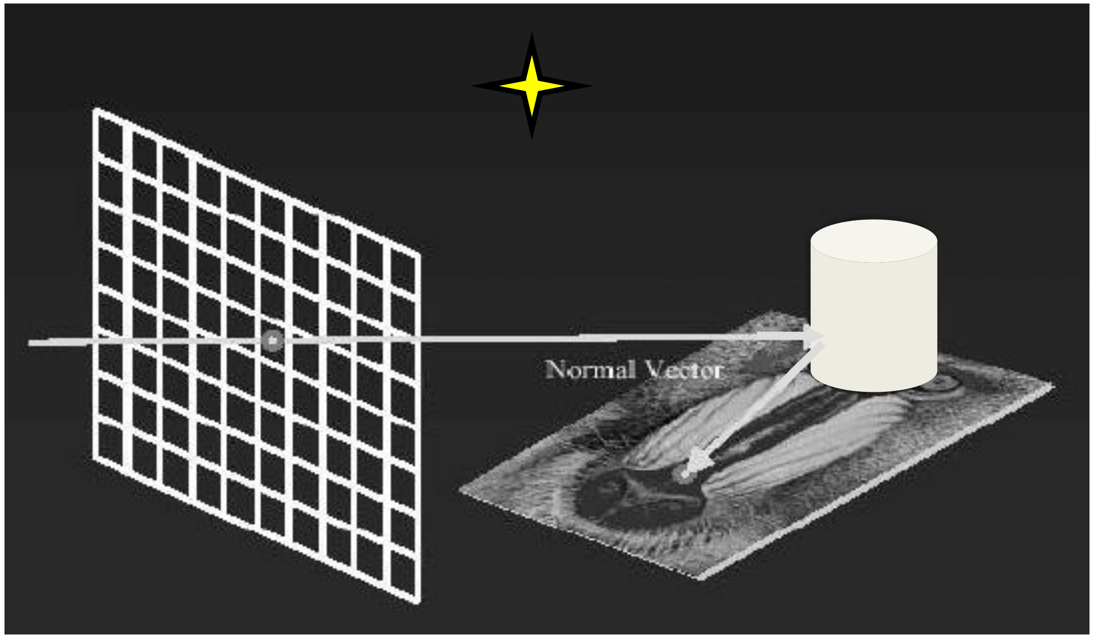
Shadow Feelers II

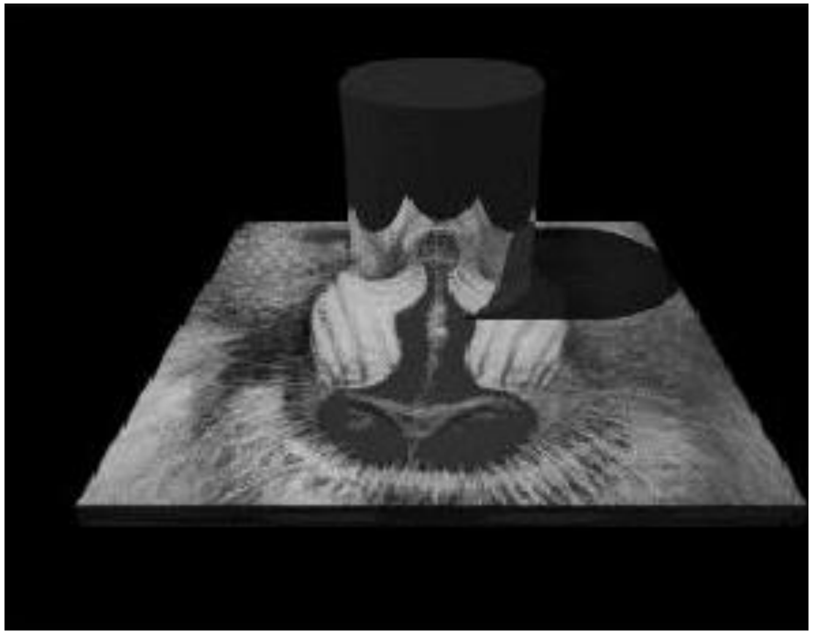




- Object between intersection point and light source: Intersection point is in the shadow

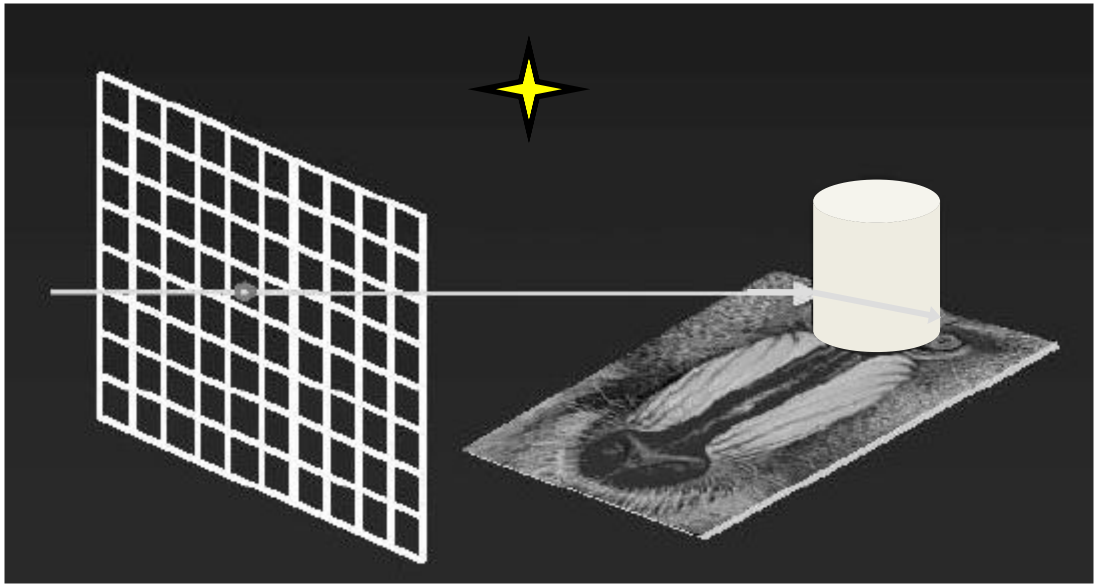
Reflection

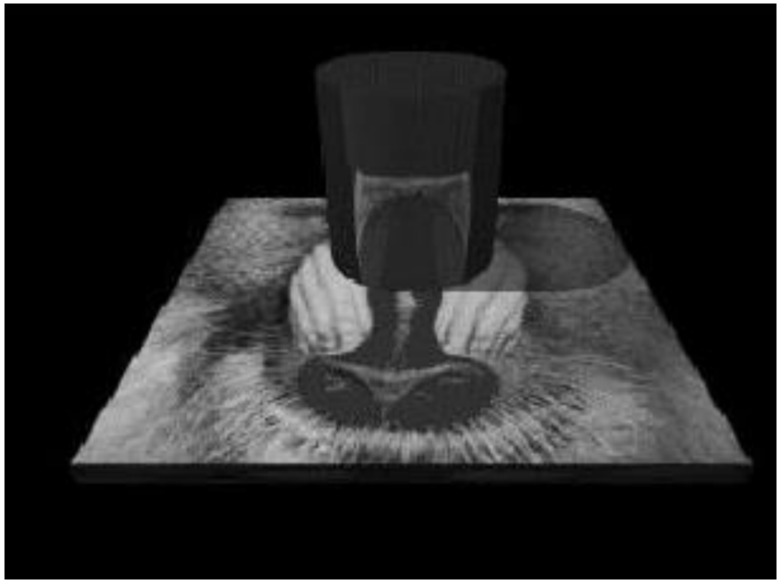




- Intersection point on mirroring object (specular reflection): Calculate and follow the reflected ray

Reflection/Deflection

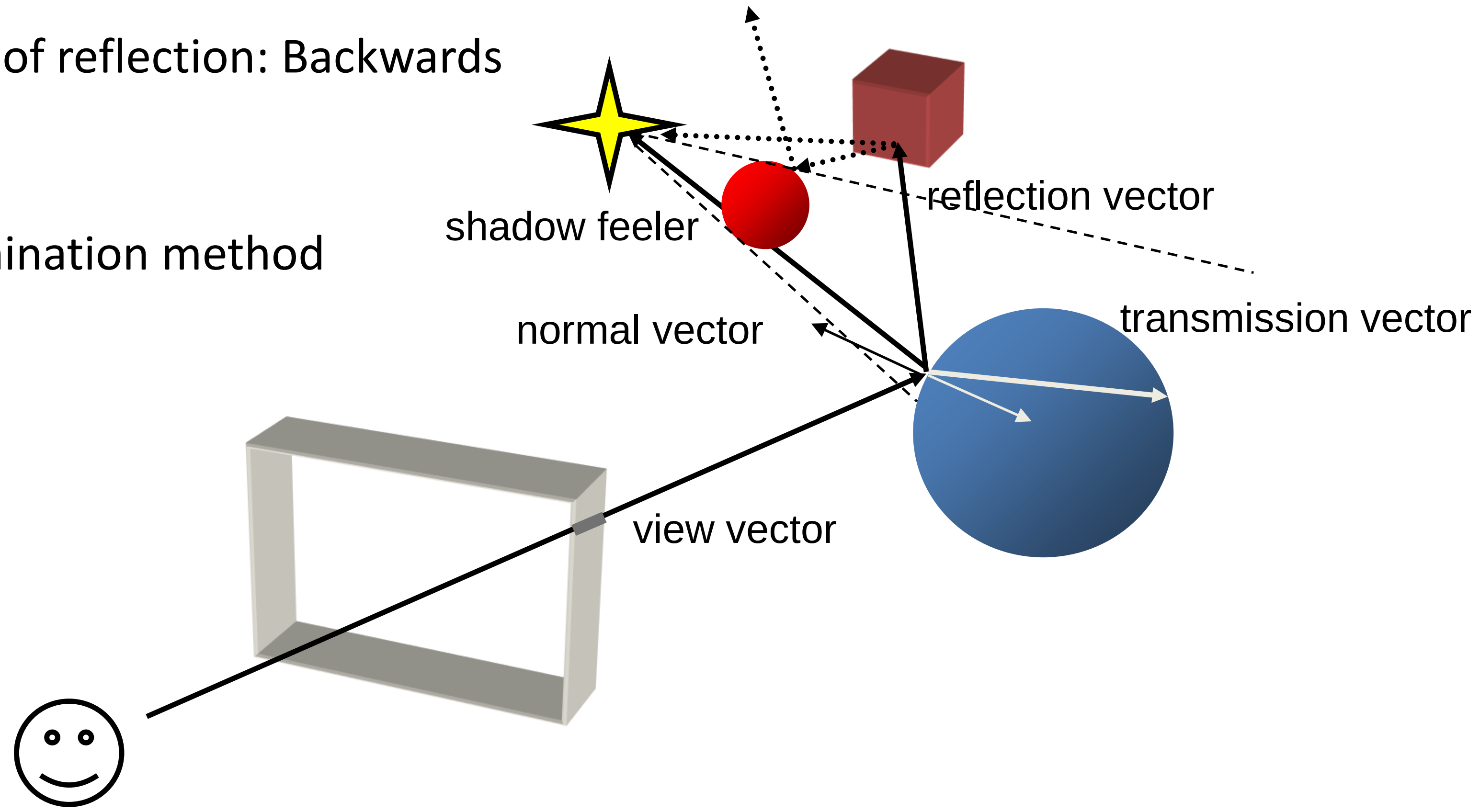




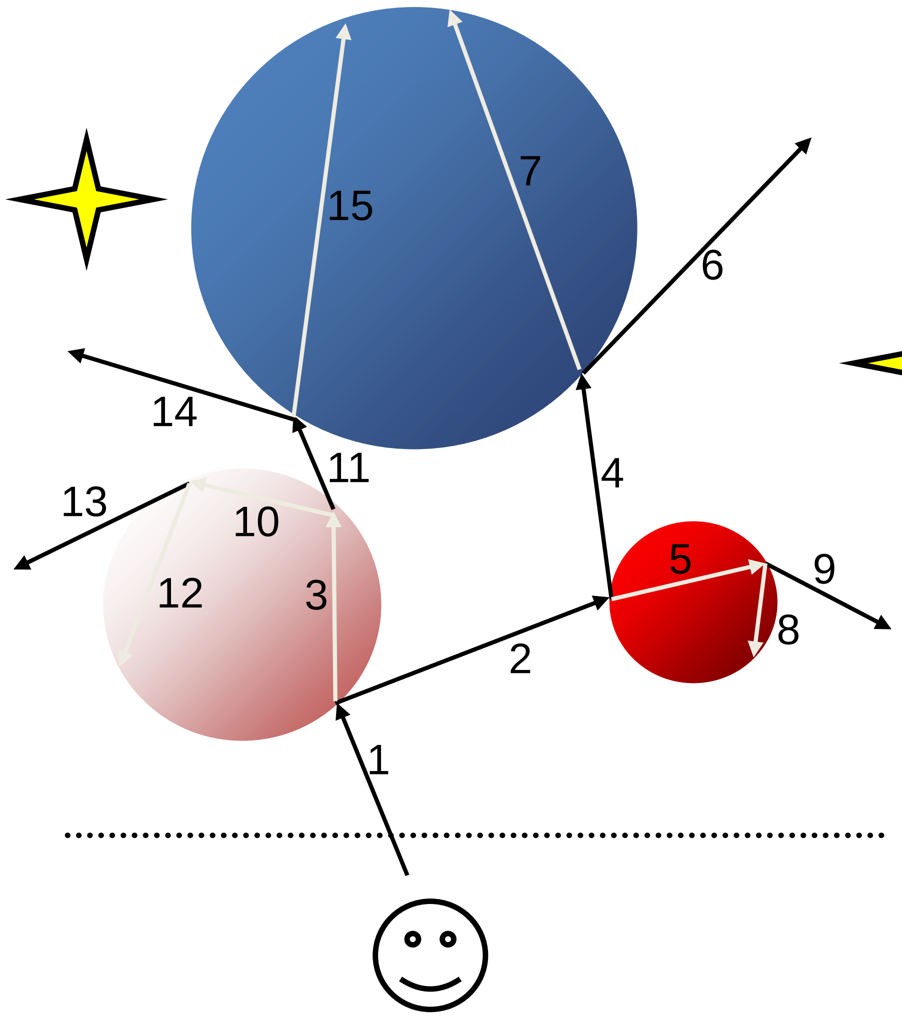
- Intersection point on a transparent object: Calculate and follow the refracted ray

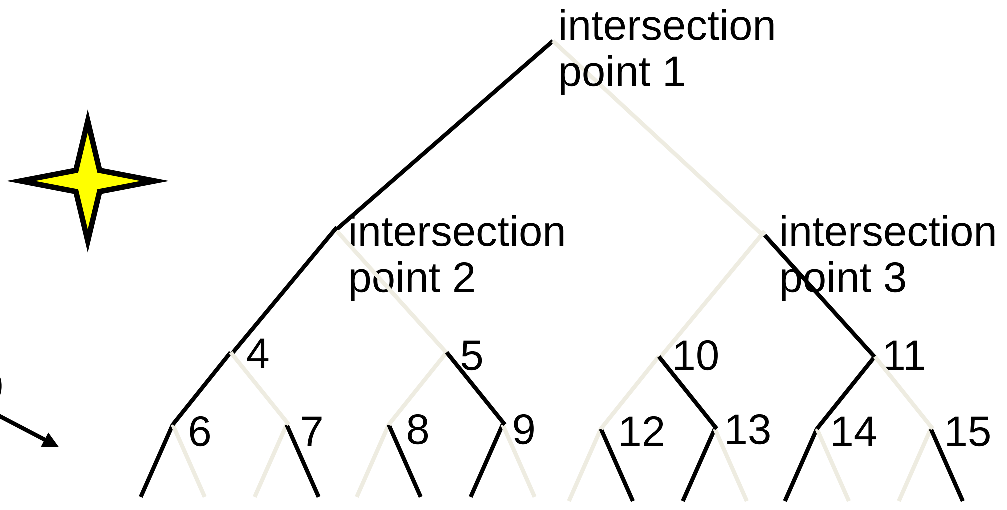
Ray Types

- Reciprocity of reflection: Backwards Ray Tracing
- Global illumination method

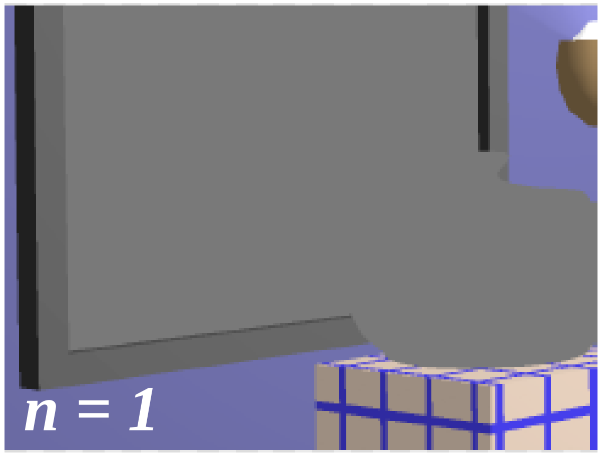


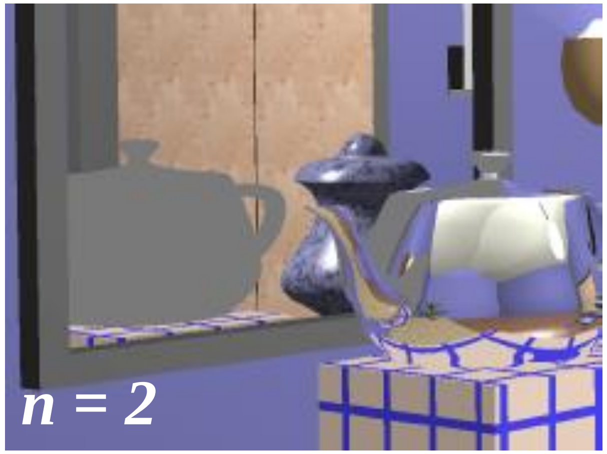
Recursion

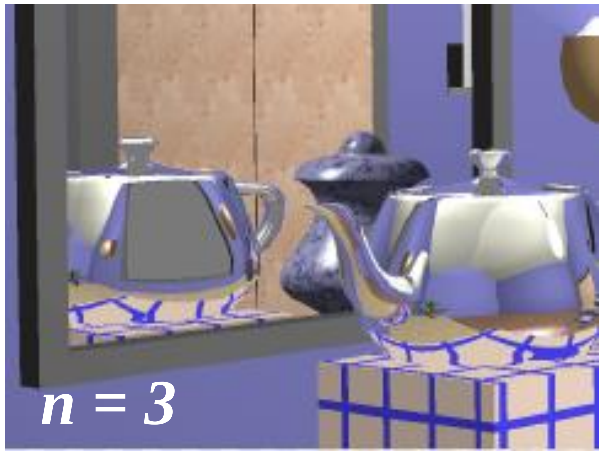

15
7
6
14
11
4
13
10
5
9
12
3
8
2
1

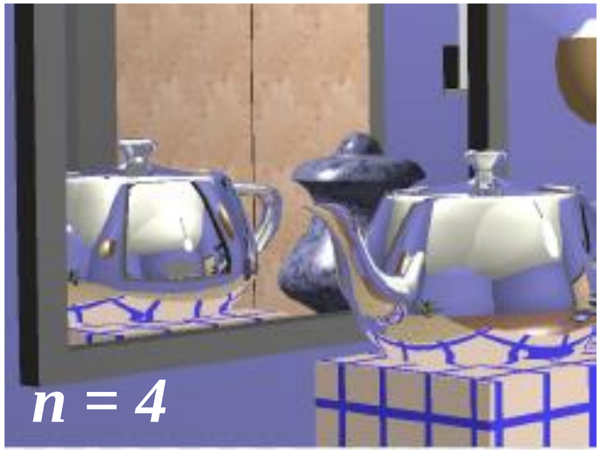

intersection point 1
intersection point 2
intersection point 3
4
5
10
11
6
7
8
9
12
13
14
15

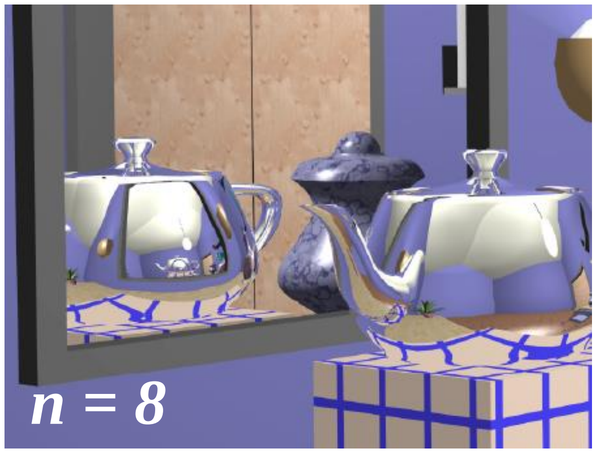
Effect on Recursion depth











The Basic Algorithm – Pseudo Code for View Rays

```
CalcImage
{
    For (y=0; y<YRES; y++)
    {
        For (x=0; x<XRES; x++)
        {
            ViewRay.Start = ViewPoint
            ViewRay.Dir = CalcViewDir(x,y,ViewPoint);
            Colour = TraceRay(ViewRay,0);
            Plot(x,y,Colour);
        }
    }
}
```

The Basic Algorithm – Pseudo Code for Ray Tracer

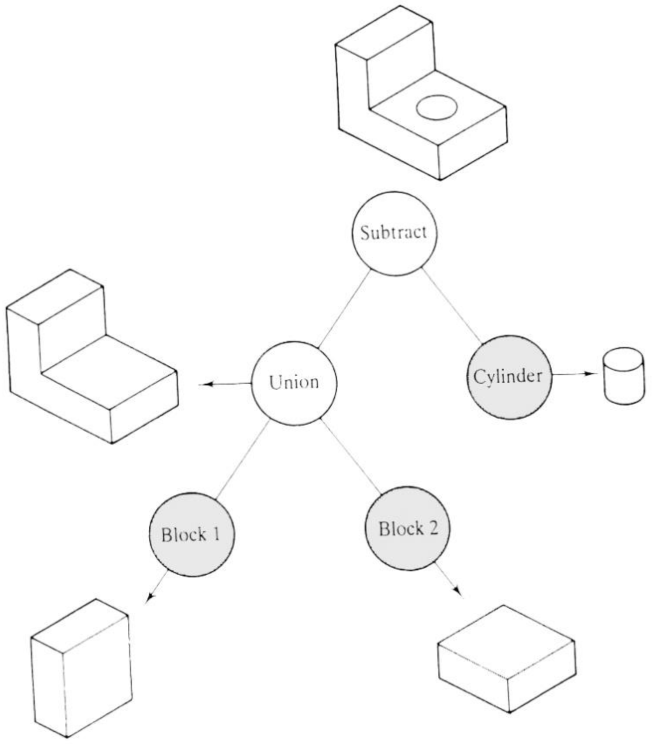
```
Colours **TraceRay** (ray: Ray, depth: int)
{
        if (depth>MAXDEPTH) return black
        else
        {
                Object, IntersectionPoint =    Schneide Strahl mit allen Objekten und
                                               ermittle nächstgelegenen Schnittpunkt; ?
                if (NoIntersection) return background_color
                else
                {
                        LocalColour = Anteile der sichtbaren Lichtquellen; ?
                        ReflectedRay = CalcReflectedRay(ray,IntersectionPoint,Object);
                        RefractedRay = CalcRefractedRay(ray,IntersectionPoint,Object); ?
                        ReflectedColour = TraceRay (ReflectedRay, depth+1);
                        RefractedColour = TraceRay (RefractedRay, depth+1);
                        return combine(LocalColour, ReflectedColour, RefractedColour);
                }
        }
}
```

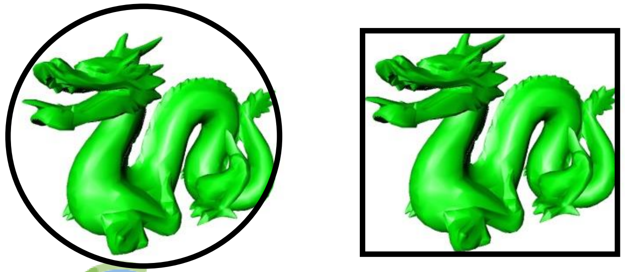
Ray Tracing – Topics

- What is Ray Tracing?
- The Basic Algorithm
- Intersections
- Light and Reflection
- Reflection- and Transmission Rays
- Optimization
    - Quality
    - Speed

Intersections – Object Geometries

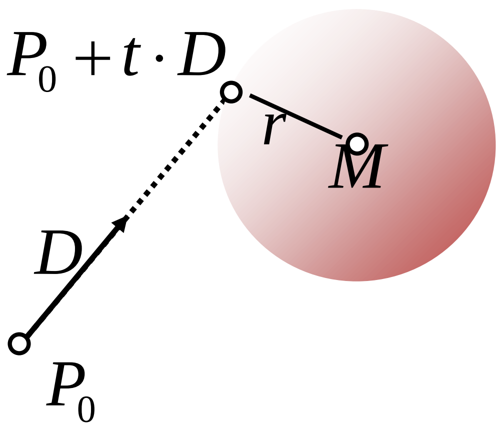
- Spheres
- Planes
- Polygons
- Polyhedrons
- Cubes
- Quadrics (cylinders, cones, ellipsoids, ...)
- Compound objects
  (e.g., Constructive Solid Geometry)
- Strategy: Approximate complex geometries by bounding volumes
  „Good“ volumes: spheres, cubes





Intersections – Sphere

$$P_S = P_0 + t \cdot D$$



$$(1)\ x_S = x_0 + t \cdot x_D,\ y_S = y_0 + t \cdot y_D,\ z_S = z_0 + t \cdot z_D$$

$$(2)\ (x_S - x_M)^2 + (y_S - y_M)^2 + (z_S - z_M)^2 = r^2$$

$$(1) \text{ in } (2):$$

$$(x_0 + t \cdot x_D - x_M)^2 + (y_0 + t \cdot y_D - y_M)^2 + (z_0 + t \cdot z_D - z_M)^2 = r^2$$

$$A \cdot t^2 + B \cdot t + C = 0 \text{ with}$$

$$A = x_D^2 + y_D^2 + z_D^2 = 1\,(D \text{ normalized})$$

$$B = 2(x_d(x_0 - x_M) + y_d(y_0 - y_M) + z_d(z_0 - z_M))$$

$$C = (x_0 - x_M)^2 + (y_0 - y_M)^2 + (z_0 - z_M)^2 - r^2$$

Intersections – Sphere (cont.)

$$t_{0/1} = \frac{-B \pm \sqrt{B^2 - 4C}}{2}$$

- Select smaller t
- If $B^2 - 4C < 0$, the ray misses the sphere
- Intersection point: $P_S = (x_S, y_S, z_S) = (x_0 + t \cdot x_D, y_0 + t \cdot y_D, z_0 + t \cdot z_D)$
- Normal vector: $N_S = \left(\frac{x_S - x_M}{r}, \frac{y_S - y_M}{r}, \frac{y_S - y_M}{r} \right)$

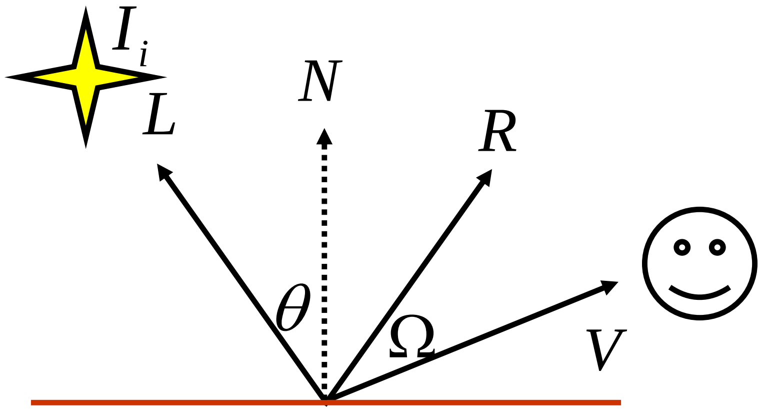
Ray Tracing – Topcis

- What is Ray Tracing?
- The Basic Algorithm
- Intersections
- Light and Reflection
- Reflection- and Transmission Rays
- Optimization
    - Quality
    - Speed

Phong Reflection Model

Linear combination from 3 components:

- Diffuse
- Specular
- Ambient



Reflection coefficients: k_d, k_s, k_a

Diffuse component: $I_d = I_i k_d \cos\theta = I_i k_d (L \cdot N)$ $\quad I_d = k_d \sum_n I_{i,n} (L_n \cdot N)$

Specular component: $I_s = I_i k_s \cos^n \Omega = I_i k_s (R \cdot V)^n$

n: Index for surface roughness

Perfect mirror: $n \rightarrow \infty$ (Ray Tracing: Recursion)

Ambient component: $I_g = I_a k_a$

Overall intensity: $I = I_a k_a + I_i (k_d (L \cdot N) + k_s (R \cdot V)^n)$

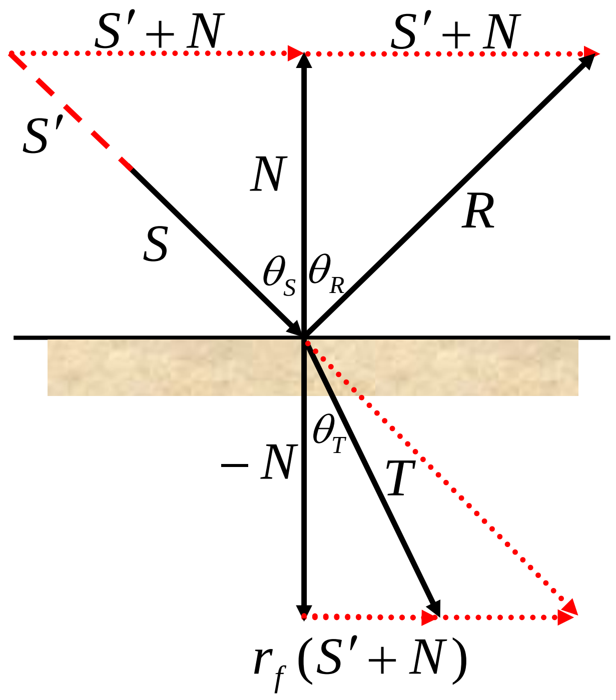
The Basic Algorithm – Pseudo Code for Ray Tracer

```
Colours **TraceRay** (ray: Ray, depth: int)
{
        if (depth>MAXDEPTH) return black
        else
        {
                Object, IntersectionPoint =     Schneide Strahl mit allen Objekten und
                                                ermittle nächstgelegenen Schnittpunkt;
                if (NoIntersection) return background_color
                else
                {
                        LocalColour = Anteile der sichtbaren Lichtquellen;
                        ReflectedRay = CalcReflectedRay(ray,IntersectionPoint,Object);
                        RefractedRay = CalcRefractedRay(ray,IntersectionPoint,Object);
                        ReflectedColour = TraceRay (ReflectedRay, depth+1);
                        RefractedColour = TraceRay (RefractedRay, depth+1);
                        return combine(LocalColour, ReflectedColour, RefractedColour);
                }
        }
}
```

?

Calculation of Reflection & Transmission Rays

- Sight (view) vector S
- reflection vector R
- Transmission vector T
- Normal vector N



- $R:\theta_S=\theta_R$ bzw. $\cos\theta_S=\cos\theta_R\ \left(-S\cdot N=N\cdot R\right)$
- $T:\dfrac{\sin\theta_S}{\sin\theta_T}=\dfrac{\eta_1}{\eta_2}$ Snell‘s Law
- Algebraic solution $R/T=\alpha S+\beta N$
- Geometrical solution

$$S'=S/_{S\cdot N}$$

$$R=S'+2N$$

$$T=r_f(S'+N)-N \quad \text{mit } r_f=\sqrt{\left(\frac{\eta_1}{\eta_2}\right)^2|S'|^2-|S'+N|^2}^{\,-1}$$

Visual Computing Institute

RWTHAACHEN UNIVERSITY

Algebraic Solution for Transmission Rays

Equation 1:

$$\begin{aligned}
\frac{\sin\theta_S}{\sin\theta_T} &= \frac{\eta_1}{\eta_2} = \eta \\
\sin\theta_S \eta &= \sin\theta_T \\
\sin^2\theta_S \eta^2 &= \sin^2\theta_T \\
\left(1-\cos^2\theta_S\right)\eta^2 &= \left(1-\cos^2\theta_T\right) \\
\left(1-\cos^2\theta_S\right)\eta^2 - 1 &= \cos^2\theta_T \\
&= \left[-N\cdot T\right]^2 \\
&= \left[-N\cdot\left(\alpha S+\beta N\right)\right]^2 \\
&= \left[\alpha\left(-N\cdot S\right)+\beta\left(-N\cdot N\right)\right]^2 \\
&= \left[\alpha\cos\theta_S-\beta\right]^2
\end{aligned}$$

Equation 2:

$$\begin{aligned}
1 &= T\cdot T \\
&= \left(\alpha S+\beta N\right)\cdot\left(\alpha S+\beta N\right) \\
&= \alpha^2\left(I\cdot I\right)+2\alpha\beta\left(I\cdot N\right)+\beta^2\left(N\cdot N\right) \\
&= \alpha^2-2\alpha\beta\cos\theta_S+\beta^2
\end{aligned}$$

Find solution for α and β and insert into $T = \alpha S + \beta N$

The Basic Algorithm – Pseudo Code for Ray Tracer

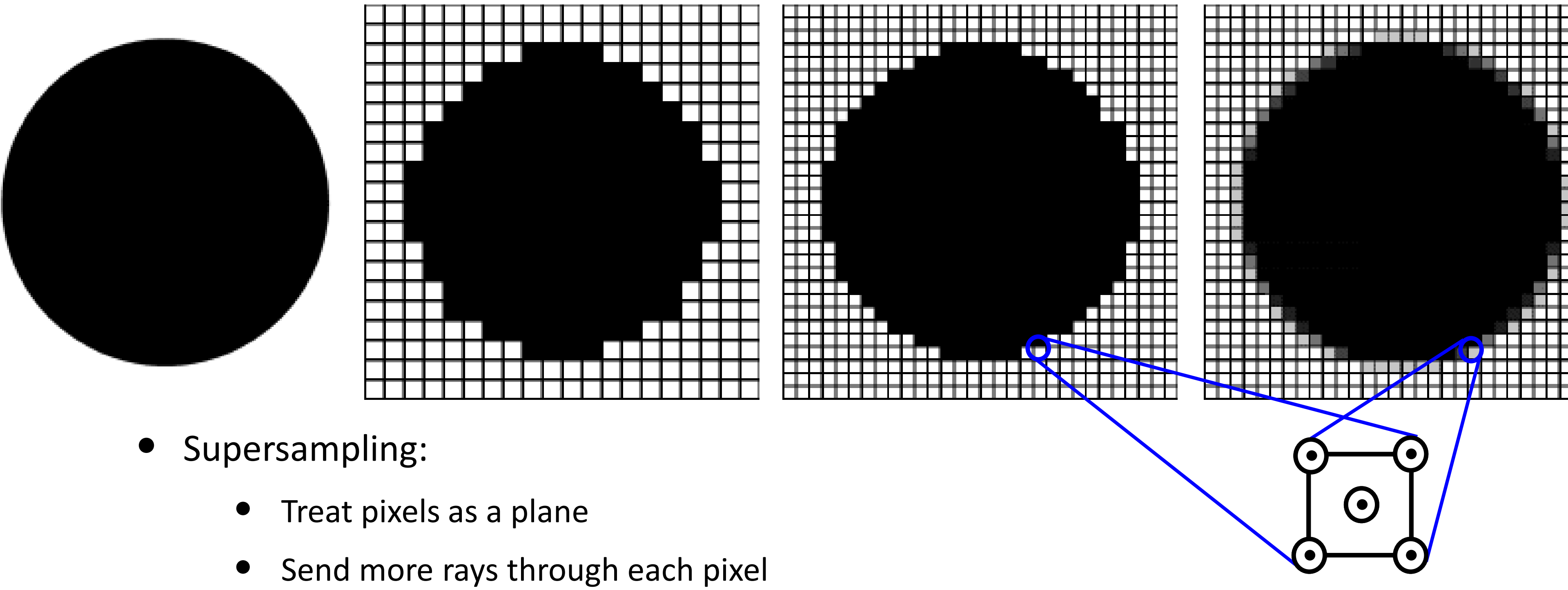
```
Colours **TraceRay** (ray: Ray, depth: int)
{
        if (depth>MAXDEPTH) return black
        else
        {
                Object, IntersectionPoint =     Schneide Strahl mit allen Objekten und
                                                ermittle nächstgelegenen Schnittpunkt;
                if (NoIntersection) return background_color
                else
                {
                        LocalColour = Anteile der sichtbaren Lichtquellen;
                        ReflectedRay = CalcReflectedRay(ray,IntersectionPoint,Object);
                        RefractedRay = CalcRefractedRay(ray,IntersectionPoint,Object);
                        ReflectedColour = TraceRay (ReflectedRay, depth+1);
                        RefractedColour = TraceRay (RefractedRay, depth+1);
                        return combine(LocalColour, ReflectedColour, RefractedColour);
                }
        }
}
```

Ray Tracing – Topics

- What is Ray Tracing?
- The Basic Algorithm
- Intersections
- Light and Reflection
- Reflection- and Transmission Rays
- Optimization
  - Quality
  - Speed

Optimization - Quality

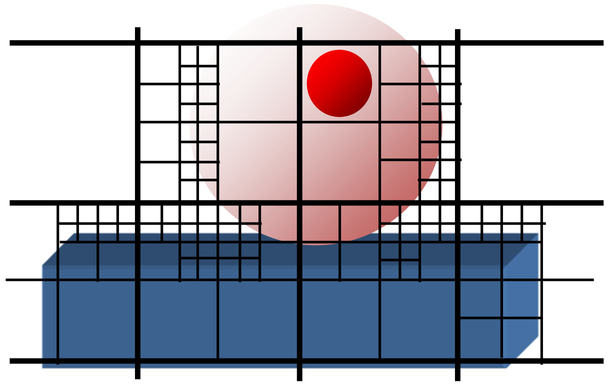
- Anti-Aliasing

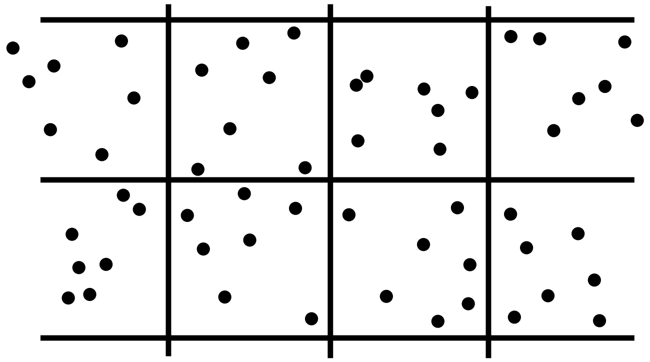


- Supersampling:
  - Treat pixels as a plane
  - Send more rays through each pixel
  - Averaging the intensities of single rays leads to pixel color

Optimization – Quality (cont.)

- Anti-Aliasing
  - Adaptive Supersampling:
    - First look at the rays at the 4 pixel corners
    - Does the intensity difference of adjacent rays exceed threshold?
    - Recursive grid refinement (Quad-Trees)
  - Stochastic Ray Tracing:
    - Irregular but uniformly dense distribution of rays





Optimization - Speed

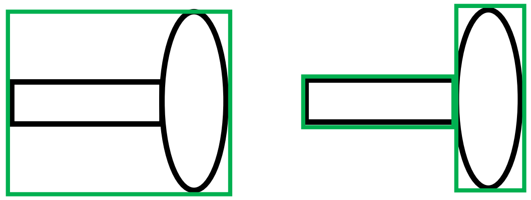
Ray Traycing is slow!

- Standard resolution SXGA (1280x1024): 1.3 Mio pixels
- Small scene with 50 objects: 65 Mio intersection points must be determined in the first step
- Algorithm complexity increases exponentially with recursion depth (in case of „specular“ objects)
- Shadow feelers cause additional intersection points (number of light sources?)
- Situation is even harder for Distributed Ray Tracing and Anti-Aliasing

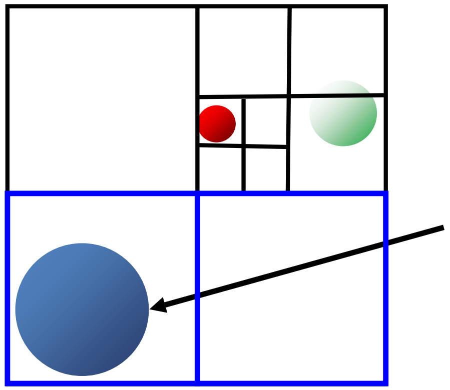
- Make use of parallelism!
- Optimize the algorithm for the calculation of single intersection points
- Reduce the number of intersection points

Optimization – Speed (II)

- Optimize the intersection point algorithm
- Bounding Volumes
  - TradeOff: Accuracy of approximation versus complexity of intersection point calculation
  - Option: Stepwise approximation



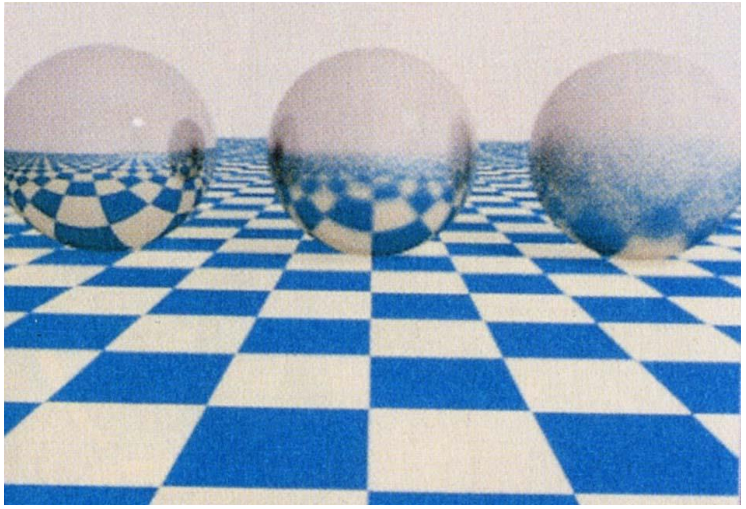
- Space subdivision
  - Octrees: Nearly the same complexity in the leaves



- Make use of the coherency for adjacent rays
  - View rays
  - Rays ending at the same plane (see polygons!)

Ray Tracing – Summary

+ Elegant, recursive algorithm for global illumination
+ Reflection, transmission, occlusion culling, hidden surface removal and shadows
+ Very realistic images if there are many specular and transparent objects in the scene



- Local modeling of diffuse reflection
- ➢ Distributed Ray Tracing
- ➢ Radiosity
- ➢ Plausibility versus authenticity: see Radiance!

- Ray Tracing is slow
- ➢ Special hardware for Ray Tracing (e.g., Advanced Rendering Technologies, Cambridge)
- ➢ Anyway: Ray Tracing may be even faster than traditional shading for scenes with millions of polygons and a lot of occlusion!

Real Time Ray Tracing

A. Dietrich, I. Wald, P. Slusallek (scene data from O. Deussen): OpenRT

28.000 sun flowers, 1 Billion polygons, 6 frames/sec (640x480 pixels), PC-Cluster (24 nodes)

Literature

Where to find the images used in the slides:

- Foley, van Dam, Feiner, Hughes: Computer Graphics: Principles and Practice. Addison Wesley, 1992
- Glassner: An Introduction to Ray Tracing, Morgan Kaufmann Publishers, 2000
- Tönnies, Lemke: 3D-Computergrafische Darstellungen, Oldenbourg Verlag, 1994
- Watt: 3D Computer Graphics, Addison Wesley, 1992
- Whitted: An improved illumination model for shaded display, Comm. of the ACM 23(6), 1980
- http://www.gris.informatik.tu-darmstadt.de/lehre/vorl_ueb/gdvII/slides/rr-bw.pdf
- http://www2.inf.fh-bonn-rhein-sieg.de/~ahinke2m/Vorlesung/CGVI03/